BBK: a simpler, faster algorithm for enumerating maximal bicliques in large sparse bipartite graphs

Alexis Baudin[1], Clémence Magnien[1], and Lionel Tabourier[1]

[1]Sorbonne Université, CNRS, LIP6, F-75005 Paris
firstname.lastname@lip6.fr

Abstract

Bipartite graphs are a prevalent modeling tool for real-world networks, capturing interactions between vertices of two different types. Within this framework, bicliques emerge as crucial structures when studying dense subgraphs: they are sets of vertices such that all vertices of the first type interact with all vertices of the second type. Therefore, they allow identifying groups of closely related vertices of the network, such as individuals with similar interests or webpages with similar contents. This article introduces a new algorithm designed for the exhaustive enumeration of maximal bicliques within a bipartite graph. This algorithm, called BBK for *Bipartite Bron-Kerbosch*, is a new extension to the bipartite case of the Bron-Kerbosch algorithm, which enumerates the maximal cliques in standard (non-bipartite) graphs. It is faster than the state-of-the-art algorithms and allows the enumeration on massive bipartite graphs that are not manageable with existing implementations. We analyze it theoretically to establish two complexity formulas: one as a function of the input and one as a function of the output characteristics of the algorithm. We also provide an open-access implementation of BBK in `C++`, which we use to experiment and validate its efficiency on massive real-world datasets and show that its execution time is shorter in practice than state-of-the art algorithms. These experiments also show that the order in which the vertices are processed, as well as the choice of one of the two types of vertices on which to initiate the enumeration have an impact on the computation time.

Keywords: Maximal biclique enumeration, bipartite graph, Bron-Kerbosch algorithm, complexity, massive real-world datasets, cliques, bicliques.

1 Introduction

Bipartite graphs are widely used to represent real-world networks [18]. They can model many systems where two different types of entities interact. They are thus widely used to describe social systems such as online platforms where users select content (watch videos, click on links, buy products, *etc.*) [36, 37], or individuals taking part in projects or events [28, 24, 11]. It is also a popular representation for biological systems [21, 29], or for ecological networks [15, 31, 9]. As with non-bipartite graphs, the identification of dense subgraphs in these networks is important for analyzing their structure and understanding their functioning [6, 33, 23]: for example, it can reveal users with common interests [27], or information about the organization of proteins [8, 32]. Also, as bipartite graphs can represent sets of items, with one type of nodes representing baskets or sets and the other the items themselves [26, 34], enumerating dense subgraphs in bipartite

graphs has a close connection with mining frequent itemsets in databases, a long-standing task in data mining [5], with various applications such as finding association rules in large databases [2].

A bipartite graph is a triplet $G = (U, V, E)$, where U and V are two sets of disjoint vertices, and E is a set of edges between elements of U and elements of V: $E \subseteq U \times V$. Graphs are undirected, so there is no distinction between an edge (u, v) and (v, u). Throughout this paper, if A is a set of vertices of $G = (U, V, E)$, then we denote by A_U the set of vertices of A that are in U, *i.e.* $A \cap U$, and A_V the set of vertices of A that are in V: $A \cap V$. A biclique of G is a set $C \subseteq U \cup V$ such that the vertices of C_U are all connected to the vertices of C_V. It is said to be maximal when it is not included in any other biclique. An example is given in Figure 1.

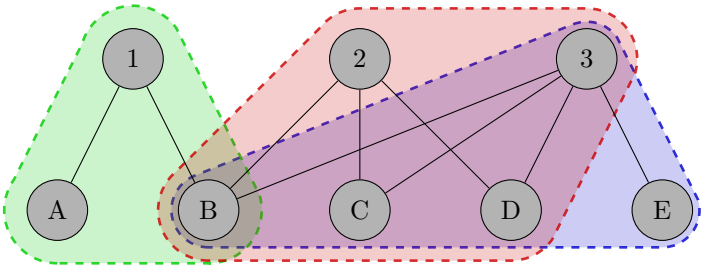

Figure 1: Example of a bipartite graph, with three maximal bicliques circled in color: $\{1, A, B\}$, $\{2, 3, B, C, D\}$ and $\{3, B, C, D, E\}$. Note that this graph has two other maximal bicliques, $\{1, 2, 3, B\}$ and $\{A, B, C, D, E\}$, not represented here for the sake of clarity.

The enumeration of bicliques, particularly in large bipartite graphs, has been the subject of much work. Recent works have notably improved the state-of-the-art of maximal biclique enumeration by adapting the famous Bron-Kerbosch algorithm [7] to this context. In this paper, we propose the BBK algorithm, which is also an adaptation of the Bron-Kerbosch algorithm to bipartite graph, but in a way that we believe to be simpler and proves to be more time-efficient than the current standards. We provide an open-access C++ implementation of this algorithm [1].

The rest of the article is organized as follows: in Section 2 we present the state of the art of maximal biclique enumeration in bipartite graphs; in Section 3 we introduce and detail our new algorithm BBK; in Section 4 we formalize complexities of this algorithm, on the one hand as a function of its input characteristics and on the other hand as a function of its output characteristics; in Section 5 we carry out an extensive experimental study of the BBK algorithm, comparing its implementation with the best in the state of the art to validate its efficiency on massive real datasets, and showing to what extent the choice of one of the two vertex sets as a starting point for BBK has an impact on computation time; finally, in Section 6 we conclude the study and present the perspectives of this work.

2 Related work

In the literature, numerous works have been devoted to enumerating maximal bicliques. Algorithms for exhaustive enumeration of maximal bicliques in bipartite graphs were introduced and experimented in the 2000s [26]. They have since been improved several times. In 2014, Damaschke *et al.* [13] developed an algorithm that improves the enumeration on bipartite graphs with a heterogeneous degree distribution, justified by the fact that this is the case for real-world graphs, with a theoretical complexity suited to the case of these particular graphs. At the same

[1] `https://gitlab.lip6.fr/baudin/bbk`

period, Zhang *et al.* [35] implemented the first adaptation of the Bron-Kerbosch algorithm [7], which is the reference algorithm for enumerating maximal cliques in non-bipartite graphs. Using this new algorithm, they handled larger datasets than what was previously possible, in particular biological datasets. Later, in 2018, Das *et al.* [14] proposed a parallel algorithm to enumerate maximal bicliques on massive datasets. In 2020, Qin *et al.* [30] developed an enumeration method which is supposed to be well suited to unbalanced graphs, however they did not provide an implementation of their algorithm. Then, Abidi *et al.* [1] improved the adaptation of the Bron-Kerbosch algorithm of Zhang *et al.* [35] by reducing the search space using a pivot, which is a technique to reduce the number of recursive calls of the main function and therefore speed up the execution. Hriez *et al.* proposed in 2021 an interesting method to achieve the enumeration [20], which consists in adding edges (a process known as graph inflation) to simplify the algorithm. However, as described the method cannot properly scale to massive graphs from the real world. Finally, Chen *et al.* [10] proposed another improvement on the algorithm of Abidi *et al.* [1], which performs much better than all the methods of the state-of-the-art. For this reason, we take Chen *et al.* method, called OOMBEA, as a reference for comparison to our own maximal biclique enumeration algorithm throughout this article.

As it plays an important role in our study, we give additional details about OOMBEA. It is inspired by the Bron-Kerbosch algorithm [7]. It is a recursive algorithm that identifies a given biclique C by its subset of C_U: the vertices in C_V are those that are neighbors to all vertices in C_U. The function then considers the *candidate* vertices $u \in U$ such that there are maximal bicliques containing $C_U \cup \{u\}$ that have not already been enumerated, and makes a recursive call on each of these vertices. Besides that, the authors also seem to take inspiration from the work of Eppstein *et al.* [16], which is one of the most efficient implementation of Bron-Kerbosch algorithm for maximal clique enumeration on real-world non-bipartite graphs. Eppstein *et al.* introduced the idea of processing vertices according to a degeneracy order [3] to reduce the number of candidate vertices on which recursive calls are made. Similarly, Chen *et al.* considered one of the two sets of vertices of the bipartite graph, which we denote by U in this paragraph, and enumerate the maximal bicliques starting from each vertex of U according to a given order which, like Eppstein *et al.*, tends to reduce the size of the set of candidate vertices. The order that OOMBEA uses is a degeneracy order of the bipartite graph projected onto U, *i.e.* the graph whose vertex set is U and two vertices are linked if they have a common neighbor in V. Chen *et al.* showed that the complexity of this algorithm is in $\emptyset n_U \cdot \zeta_U \cdot m \cdot 2^{\zeta_U}$, where n_U is the number of vertices in U, ζ_U is the degeneracy of the graph projected onto U and m is the number of edges in G. They also expressed the complexity of OOMBEA in terms of the number of maximal bicliques $\mathcal{B}$ as $\emptyset \zeta_U \cdot m \cdot \mathcal{B}$. Note that even for a sparse graph, the graph projected onto U can be dense, and therefore can have a high ζ_U value.

3 New maximal biclique enumeration algorithm

In this section, we use a graph-inflation process to transform the bipartite graph and then apply directly the Bron-Kerbosch algorithm [7] to enumerate maximal bicliques. We can then use the version of this algorithm from Eppstein *et al.* [16], which processes the vertices in a specific order to improve the computation time for massive real-world graphs. To do this, we define what we call a *bidegeneracy order* of vertices, adapted from the degeneracy order proposed by Eppstein *et al.* This provides a simple way to perform the maximal biclique enumeration. However, this first approach is inefficient, so we use properties of neighborhood in bipartite graphs to enhance its performances in the BBK algorithm.

3.1 Clique-extended graph of a bipartite graph

To extend the enumeration of maximal cliques in graphs to the enumeration of maximal bicliques in bipartite graphs, we define the *clique-extended graph* of a bipartite graph by adding edges between all the vertices of U, and between all the vertices of V. We call this graph G^C and define it formally below.

Definition 3.1 (Clique-extended graph of a bipartite graph)**.** *The* clique-extended graph of the bipartite graph G *is the graph* $G^C = (U \cup V, E^C)$*, where:*

$$E^C = E \cup \{(u_1, u_2) \in U^2 \mid u_1 \neq u_2\} \cup \{(v_1, v_2) \in V^2 \mid v_1 \neq v_2\}.$$

This clique-extended graph induces a notion of neighborhood, which we call *clique-extended neighborhood*, noted N^C:

Definition 3.2 (Clique-extended neighborhood of a vertex)**.** *Let* $x \in U \cup V$*. The* clique-extended neighborhood of x *corresponds to the neighbors of* x *in the clique-extended graph* G^C*. It is denoted by* $N^C(x)$*:*

$$N^C(x) = \begin{cases} N(x) \cup U \setminus \{x\} \text{ if } x \in U \\ N(x) \cup V \setminus \{x\} \text{ if } x \in V. \end{cases}$$

The clique-extended graph has a particular property, that we exploit for the BBK algorithm: a set of vertices that forms a (maximal) clique of G^C equivalently forms a (maximal) biclique of G. This result was introduced by Gély *et al.* [17]:

Theorem 3.1. *Let* $G = (U, V, E)$ *be a bipartite graph. Then the maximal cliques of* G^C *correspond to the maximal bicliques of* G*:*

$$C \text{ is a maximal clique of } G^C \Leftrightarrow C \text{ is a maximal biclique of } G.$$

This theorem induces a direct method for enumerating the maximal bicliques of a bipartite graph: it is sufficient to enumerate the maximal cliques of its extended graph. Algorithm 1 proposes the pseudocode for this method: it follows the Eppstein *et al.* algorithm [16].

3.2 bbk: a new algorithm for maximal biclique enumeration

Algorithm 1 is straightforward, but it cannot be used in practice for graphs containing many vertices, as the sets of candidates P_i defined at Line 2 can be larger than U or V. To overcome this issue and provide an algorithm usable efficiently on sparse, massive, bipartite graphs, we develop a revised version which takes advantage of the bipartite nature of the graph. This allows to reduce the size of the candidate sets and to use N instead of N^C in the main function, which induces much fewer neighbors. In addition, we refine the biclique maximality test to perform it earlier in the process; we also limit the enumeration to bicliques containing a vertex u for each $u \in U$ instead of browsing all $U \cup V$; finally, we order the vertices to improve the efficiency of the enumeration by introducing the notion of a *bidegeneracy order*. Each of these points is detailed in the rest of this section.

We call this new algorithm BBK, and its pseudocode is given in Algorithm 2. Let us first summarize the workings of BBK before going into its details:

- Lines 1 to 4 give a more efficient way of initializing biclique enumerations on each vertex, to be compared with Lines 1 to 4 of Algorithm 1;

Algorithm 1: Enumerate the maximal bicliques using the extended graph.

Input: Bipartite graph $G = (U, V, E)$.
Output: Set of maximal bicliques of G.

```
1  for each x_i in a degeneracy order x_1, x_2, ..., x_n of U ∪ V do
2      P_i ← N^C(x_i) \ {x_1, ..., x_{i-1}}
3      X_i ← N^C(x_i) ∩ {x_1, ..., x_{i-1}}
4      BronKerbosch({x_i}, P_i, X_i)
5  Function BronKerbosch(R, P, X):
6      if P ∪ X = ∅ then
7          return R maximal biclique
8      p ← pivot in P ∪ X
9      for x ∈ P \ N^C(p) do
10         BronKerbosch(R ∪ {x}, P ∩ N^C(x), X ∩ N^C(x))
11         P ← P \ {x}
12         X ← X ∪ {x}
```

- Lines 6 to 9 are an improvement on the maximality test for bicliques performed in Lines 6 to 7 of Algorithm 1;
- finally, Lines 10 to 26 are equivalent to Lines 8 to 12 of Algorithm 1, but adapted to use the neighborhood N in the bipartite graph, instead of N^C.

3.2.1 Lines 1 to 4: efficient initialization of the biclique enumeration

Set of vertices for the initialization The initialization of calls to `BronKerbosch` performed at Line 4 of Algorithm 1 can be improved by performing the following two operations.

Firstly, it is not necessary to enumerate the set of maximal bicliques that contain x for each vertex $x \in U \cup V$. Instead, we can simply list the maximal bicliques that contain a vertex $u \in U$. Indeed, V is the only biclique that contains no vertices in U and that can be maximal; it can therefore be added to the enumeration outside the core of the algorithm. Thus, in Algorithm 2, the loop starting at Line 1 is only performed on U. Note that this idea has already been used by Chen *et al.* [10] in OOMBEA.

Secondly, to enumerate the maximal cliques in a graph, the Bron-Kerbosch algorithm uses the fact that the neighbors of u are the vertices belonging to a clique which contains u. In a bipartite graph, this is not the case: if $u \in U$, the set of vertices that are in a biclique C which contains u, when $C_V \neq \emptyset$, are *the vertices of* $N(u) \cup N_2(u)$ where $N_2(u)$ is the set of neighbors of u's neighbors excluding u itself. Note that this observation was made by Hermelin and Manoussakis [19]. Therefore, we formalize below this particular neighborhood of u as it plays an important role in the description of the BBK algorithm.

Definition 3.3 (Projection-extended neighborhood)**.** *Let $u \in U$. We call the vertices of $N(u) \cup N_2(u)$ the* projection-extended neighborhood of u *and denote it by $N^P(u)$, where $N_2(u)$ is the set of neighbors of u's neighbors excluding u itself.*

We use this projection-extended neighborhood by searching, for each vertex $u \in U$, the maximal bicliques that contain u among the vertices of the set $N^P(u)$. Thus, the sets $N^C(u_i)$

Algorithm 2: BBK: Bron-Kerbosch adapted to maximal biclique enumeration.

Input: Bipartite graph $G = (U, V, E)$.
Output: Set of maximal bicliques of G.

```
// More efficient initialization:
```
for each u_i *in a bidegeneracy order* $u_1, u_2, \ldots, u_n$ *of* U **do**
  $P_i \leftarrow N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}$
  $X_i \leftarrow N^P(u_i) \cap \{u_1, \ldots, u_{i-1}\}$
  `BipBronKerbosch(`$\{u_i\}$`,` P_i`,` X_i`)`
Function `BipBronKerbosch(`R`,` P`,` X`):`
  `// Maximality test:`
  if *(*$P_U = \emptyset$ **or** $P_V = \emptyset$*)* **and** $X = \emptyset$ **then**
    return $R \cup P$ maximal biclique
  if *(*$P_U = \emptyset$ **and** $X_V \neq \emptyset$*)* **or** *(*$P_V = \emptyset$ **and** $X_U \neq \emptyset$*)* **then**
    return
  `// BronKerbosch adapted operations to leverage bipartite nature of the graph:`
  $p \leftarrow$ pivot in $P \cup X$
  if $p \in U$ **then**
    $Q \leftarrow P_V \setminus N(p)$
  else
    $Q \leftarrow P_U \setminus N(p)$
  if $p \in P$ **then**
    $Q \leftarrow \{p\} \cup Q$
  for $x \in Q$ **do**
    if $x \in U$ **then**
      $P_x \leftarrow (P_U \setminus \{x\}) \cup (P_V \cap N(x))$
      $X_x \leftarrow X_U \cup (X_V \cap N(x))$
    else
      $P_x \leftarrow (P_V \setminus \{x\}) \cup (P_U \cap N(x))$
      $X_x \leftarrow X_V \cup (X_U \cap N(x))$
    `BipBronKerbosch(`$R \cup \{x\}$`,` P_x`,` X_x`)`
    $P \leftarrow P \setminus \{x\}$
    $X \leftarrow X \cup \{x\}$

at Line 1 of Algorithm 1 are replaced by the sets $N^P(u_i)$ at Line 1 of Algorithm 2, which are much smaller in practice (see Section 5).

Note that this reasoning ignores the biclique U which is the only one which does not contain any vertex in V. As above, this biclique can be added to the enumeration outside the core of the algorithm if it is maximal.

Bidegeneracy order of vertices Eppstein *et al.* [16] have shown that the order of vertices has a significant impact on the enumeration efficiency on non-bipartite real-world graphs. They use a degeneracy order to reduce the maximum size of the candidate vertex sets P_i on which recursive calls are made. We extend this concept by introducing the notion of a *bidegeneracy order* on U:

Definition 3.4 (Bidegeneracy order of U)**.** *A* bidegeneracy order of U *is an order* $u_1, \ldots, u_n$ *such that* u_i *is a vertex of* U *of minimal number of projection-extended neighbors in the subgraph induced by the vertices* $u_i, u_{i+1}, \ldots, u_n$ *and their neighbors. In other words, for all* $i \in \{1, \ldots, n\}$*,*

$$u_i = \underset{u \in \{u_i, \ldots, u_n\}}{\operatorname{argmin}} \left(|N^P(u) \setminus \{u_1, \ldots, u_{i-1}\}| \right).$$

Such an order is obtained by iteratively selecting an unselected vertex $u \in U$ that minimizes $N^P(u)$, then updating the sets in N^P by deleting u. The objective of using a bidegeneracy order of U is to reduce the maximum size of the candidate sets $P_i = N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}$ on which the enumeration is performed at Line 4 of Algorithm 2. To quantify this maximum size, we introduce below the notions of the *bidegeneracy of a vertex* and the *bidegeneracy of a set of vertices*.

Definition 3.5 (Bidegeneracy of a vertex)**.** *The* bidegeneracy of a vertex $u \in U$ *is the maximum value* b *such that there exists* $U' \subseteq U$ *with* $u \in U'$ *verifying* $\forall x \in U', \left|N^P(x) \cap (U' \cup V)\right| \geq b$*. We denote it by* $b(u)$*. If* $u \in V$*, its bidegeneracy is defined symmetrically by inverting* U *and* V*.*

Definition 3.6 (Bidegeneracy of U and V)**.** *The bidegeneracy of* U*, denoted by* b_U*, is the maximum bidegeneracy of the vertices of* U*. The bidegeneracy of* V*, denoted* b_V*, is defined similarly on* V*.*

For example, let us consider the graph of Figure 1 with $U = \{A, B, C, D, E\}$ and $V = \{1, 2, 3\}$. In this example, $N^P(A) = \{1, B\}$, so if we set $U' = \{A, B\}$, then $\forall x \in U'$, $|N^P(x) \cap (U' \cup V)| \geq 2$; moreover there is no set which yields a larger value, thus $b(A) = 2$. We can show similarly that the bidegeneracy of B, C, D and E is 4, thus $b_U = 4$.

Thanks to the use of a bidegeneracy order, we can show that the size of a candidate set $P_i = N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}$ at Line 2 of Algorithm 2 is at most $b(u_i)$ and therefore the maximum size of this set over all vertices in U is b_U. Indeed, when u_i is selected following such an order, it is a vertex of minimal number of projection-extended neighbors in the subgraph induced by $u_i, u_{i+1}, \ldots, u_n$ and their neighbors. In other words, if we set $U' = \{u_i, \ldots, u_n\}$, then $\forall u \in U', |N^P(u) \setminus \{u_1, \ldots, u_{i-1}\}| = |N^P(u) \cap (U' \cup V)| \geq |N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}|$. Since the bidegeneracy of u_i is the largest value over all sets U' satisfying the inequality above, we obtain that $b(u_i) \geq |N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}|$. This improvement is efficient in that the bidegeneracy is much smaller in practice than the maximum value of $|N^P(u)|$ for $u \in U$ (see Section 5). This is particularly important as the enumeration within one of these sets is exponential in the size of that set, as detailed in Section 4.

3.2.2 Lines 6 to 9: improving the biclique maximality test

The maximality test performed at Line 6 of Algorithm 1 can be performed earlier by taking into account the bipartite nature of the graph. Indeed, if $P_U = \emptyset$, then there are two cases which require no recursive call on any vertex of P_V and that can be tested in constant time:

- If $X = \emptyset$, then $R_U \cup R_V \cup P_V$, which is a biclique by construction, is maximal and can therefore be output without making further recursive calls (Lines 6 - 7).
- If $X_V \neq \emptyset$, then X_V cannot be modified again, as X_V is only modified at Lines 20 when a vertex of P_U is added to the clique under construction. Therefore, $P_U = \emptyset$ and $X_V \neq \emptyset$ for all the subcalls launched at Line 24, and then none of them can lead to a maximal biclique (which would require X_V to be empty). Thus, the call can be stopped there (Lines 8 - 9).

The same observations can be made when swapping the roles of U and V. Altogether, Lines 6 to 7 in Algorithm 1 are replaced by Lines 6 to 9 in Algorithm 2.

3.2.3 Lines 10 to 26: using N instead of N^C

The neighborhoods N^C in Algorithm 1 are usually too large to be processed efficiently on massive graphs. Fortunately, it is not necessary to store and manipulate them in practice. Indeed, it is possible to compute the sets handled in Lines 8 to 12 of Algorithm 1 by considering only the bipartite neighborhood N. In this sense, we show that Lines 10 to 26 of Algorithm 2 are equivalent to those lines.

When $x \in U$, the set $P \cap N^C(x)$ (Line 10 of Algorithm 1) is equal to $P \cap (N(x) \cup U \setminus \{x\})$, thus, it is equal to $(P_U \setminus \{x\}) \cup (P_V \cap N(x))$. Symmetrically, when $x \in V$, $P \cap N^C(x)$ is equal to $(P_V \setminus \{x\}) \cup (P_U \cap N(x))$. The same applies to $X \cap N^C(x)$, following the same reasoning. This leads to the sets P_x and X_x defined within the loop starting at Line 17 of Algorithm 2.

We can apply the same reasoning to the pruning of the pivot occurring at Line 9 of Algorithm 1. Indeed, if $p \in U$, $P \setminus N^C(p) = P \setminus (N(p) \cup U \setminus \{p\}) = (P_U \setminus (U \setminus \{p\})) \cup (P_V \setminus N(p)) = \{p\} \cup (P_V \setminus N(p))$ if $p \in P$, and $P \setminus N^C(p) = P_V \setminus N(p)$ if $p \notin P$ (Lines 12 and 16). Similarly, if $p \in V$ then $P \setminus N^C(p) = \{p\} \cup (P_U \setminus N(p))$ if $p \in P$, and $P \setminus N^C(p) = P_U \setminus N(p)$ if $p \notin P$ (Lines 14 and 16). Note in particular that the pivot prunes all vertices (except itself) on its own side, in addition to removing vertices from $N(p)$ in the other side, which is more efficient than only pruning the vertices of $N(p)$.

As with the maximal clique enumeration in non-bipartite graphs, the pivot is chosen at Line 10 to maximize the number of vertices pruned, *i.e.* to minimize the size of Q.

4 Complexity of bbk algorithm

In this section, we express the complexity of Algorithm BBK as a function of its input and output characteristics (resp. Section 4.1 and Section 4.2). To do so, we use that Algorithm BBK has been inspired by Eppstein *et al.* work on maximal clique enumeration [16]. Thus, we can derive its complexity as a function of its input following an approach similar to theirs, and the complexity as a function of its output following later works [12, 4].

In what follows, we denote by d_U the maximum degree of a vertex of U, and by d_V the maximum degree of a vertex of V.

4.1 Complexity as a function of input characteristics

Before expressing the complexity as a function of the input in Theorem 4.1, let us first introduce the following preliminary Lemma 4.1.

Lemma 4.1. *The bidegeneracy of U is larger than the maximum degrees of U and V:*

$$d_U \leq b_U \text{ and } d_V \leq b_U.$$

Proof. Let $u \in U$ be a vertex of degree $d(u) = d_U$, and consider $U' = \{u\}$, then $\forall x \in U',\ |N^P(x) \cap (U' \cup V)| = d_U$. Thus, $b(u) \geq d_U$, hence $b_U \geq d_U$.

Now let $v \in V$ be a vertex of degree $d(v) = d_V$, and consider $U' = N(v)$, then since each vertex of $N(v)$ is adjacent to v, we deduce that $\forall x \in U',\ |N^P(x) \cap (U' \cup V)| \geq d_V$. Thus, for any $u \in U'$, $b(u) \geq d_V$ hence $b_U \geq d_V$. □

Theorem 4.1. *The complexity of Algorithm 2* BBK *is in $\emptyset(n_U + m) \cdot b_U \cdot 3^{b_U/3}$, where n_U is the number of vertices in U, m is the number of edges and b_U is the bidegeneracy of U.*

Proof. The algorithm first computes a bidegeneracy order. This requires, for each vertex $u \in U$, to calculate its projection-extended neighborhood $N^P(u)$ in $\emptyset d_U \cdot d_V$. Then, the bidegeneracy order is computed by iteratively taking a vertex u of minimal $|N^P(u)|$ and decreasing by 1 the value of $|N^P(u')|$ for each $u' \in N^P(u) \cap U$, and there are at most $d_U \cdot d_V$ such nodes. The whole procedure is thus carried out in $\emptyset n_U \cdot d_U \cdot d_V$, that is in $\emptyset n_U \cdot {b_U}^2$ according to Lemma 4.1.

Then, for each vertex $u_i \in U$ in the resulting order, the algorithm enumerates the maximal bicliques containing u_i and no vertex preceding it in the order, using the `BipBronKerbosch` function at Line 4. It begins with computing the sets $P_i = N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}$ and $X_i = N^P(u_i) \cap \{u_1, \ldots, u_{i-1}\}$ at Lines 2 and 3, in $\emptyset d_U \cdot d_V$. Then, to evaluate the cost of the call to `BipBronKerbosch` at Line 4, we can here apply the complexity expression of the Eppstein *et al.* algorithm [16] in the case of a non-bipartite graph, as it can be noticed that the cost of the operations performed are the same. To do so, we can use their Lemma 3.6 where the authors show that given $c \geq |P_i|$, this call is done in $\emptyset(c + |X_i|) \cdot 3^{c/3}$, when choosing a pivot that maximizes the number of cut vertices. So here the complexity of the call is in $\emptyset(b(u_i) + |X_i|) \cdot 3^{b(u_i)/3}$, as by definition of a bidegeneracy order, we know that the size of the candidate set P_i is at most $b(u_i)$ (see Section 3.2.1). Therefore, the loop at Line 1 is done in $\emptyset \sum_{i=1}^{n_U} (b(u_i) + |X_i|) \cdot 3^{|P_i|/3}$, that is in $\emptyset\left(n_U \cdot b_U + \sum_{i=1}^{n_U} |X_i|\right) \cdot 3^{b_U/3}$. Now, $\sum_{i=1}^{n_U} |X_i| \leq \sum_{u \in U} |N^P(u)| \leq \sum_{u \in U} |N(u)| + \sum_{u \in U} |N_2(u)|$ (see Definition 3.3 that defines N^P). Besides, $\sum_{u \in U} |N(u)| = m$ and $\sum_{u \in U} |N_2(u)| \leq \sum_{u \in U} \sum_{v \in N(u)} d(v) \leq \sum_{u \in U} (d_V \cdot |N(u)|) = d_V \cdot m$. Thus, $\sum_{i=1}^{n_U} |X_i|$ is in $\emptyset d_V \cdot m$, that is in $\emptyset b_U \cdot m$ according to Lemma 4.1. Then, the loop at Line 1 is done in $\emptyset(n_U + m) \cdot b_U \cdot 3^{b_U/3}$.

Finally, the BBK algorithm runs in $\emptyset(n_U + m) \cdot b_U \cdot 3^{b_U/3} + n_U \cdot {b_U}^2$. In addition, for any integer $k \geq 0$, $k^2 \leq k \cdot 3^{k/3}$, so in particular ${b_U}^2 \leq b_U \cdot 3^{b_U/3}$ which leads to the complexity expression in the statement. □

This complexity should be compared with that of Chen *et al.*'s OOMBEA algorithm [10]. They show that it runs in $\emptyset m \cdot n_U \cdot \zeta_U \cdot 2^{\zeta_U}$, where ζ_U, called *unilateral coreness*, is the degeneracy of the graph projected onto U, and m is the number of edges in G. The bidegeneracy and the unilateral coreness are closely related concepts, with $b_U \geq \zeta_U$. The factor $(n_U + m) \cdot b_U \cdot 3^{b_U/3}$

in our complexity therefore corresponds to the factor $m \cdot \zeta_U \cdot 2^{\zeta_U}$ in the OOMBEA algorithm complexity, whereas this second complexity features an additional factor n_U.

Several practical observations can be made about the complexity expression of Theorem 4.1. First, the bidegeneracy of U plays a central role in the complexity, due to the exponential factor in b_U, which also points out the benefit of using a bidegeneracy order. Indeed, this order makes it possible to bound by b_U the maximum size of a candidate set $P_i = N^P(u_i) \setminus \{u_1, \ldots, u_{i-1}\}$, while a random order would lead to bound it by $d_U + d_{2U}$, where d_{2U} is the maximum degree of the graph projected onto U. Furthermore, we show in Section 5 (see Table 2) that the maximal bidegeneracy is in practice close to its optimal value, *i.e.* the maximal degree of the graph (Lemma 4.1). In other words, without the bidegeneracy order, there would be an additional d_{2U} in the bound, and this would be of less benefit as the projected degree is much larger in practice than the bidegeneracy.

It should be also noted that this complexity in $\emptyset n_U \cdot b_U \cdot 3^{b_U/3}$ is more precisely in $\emptyset \sum_{i=1}^{n_U} (b(u_i) + |X_i|) \cdot 3^{b(u_i)/3}$, as seen in the proof of Theorem 4.1. It is a relevant point, as $b(u_i)$ of most vertices $u_i \in U$ is well below the maximum value b_U in massive real-world networks, as can be seen in Table 2: the average bidegeneracy in real-world datasets is usually much lower than the maximum bidegeneracy. However, the exponential factor $3^{b_U/3}$ in the complexity formula can be high on real data, even in cases where BBK exhibits good performance in practice (see Section 5). That is because this worst case complexity can be far from the actual running time of the algorithm, which is why we develop in the following subsection a complexity expression as a function of the algorithm output characteristics.

4.2 Complexity as a function of output characteristics

Now, we formulate the complexity of Algorithm 2 BBK as a function of its output characteristics, precisely:

- $\boldsymbol{\mathcal{B}}$: the number of maximal bicliques in G;
- $\boldsymbol{q}$: the maximal size of a non-trivial biclique (meaning that $C_U \neq \emptyset$ and $C_V \neq \emptyset$);
- $\boldsymbol{d}$: the maximal degree in G;
- $\boldsymbol{d_U}$: the maximal degree of vertices in U, *i.e.* $d_U = max_{u \in U}\{|N(u)|\}$;
- $\boldsymbol{d_{2U}}$: the maximal degree of the graph projected onto U, *i.e.* $d_{2U} = max_{u \in U}\{|N_2(u)|\}$

To do this, we consider the trees of recursive calls made by function `BipBronKerbosch` within Algorithm 2. The initializing call of this function is made at Line 4, and recursive calls are made at Line 24. The internal nodes of these trees correspond to calls for which the set of vertices Q on which iterates the loop on Line 17 is not empty, *i.e.* they generate other child calls, while the leaves correspond to calls that generate no other.

Inspired by the work of Conte *et al.* [12], we focus in what follows on the leaves of these call trees, which we separate into two categories: those that output a maximal clique and those that do not. The latter correspond to unnecessary computations, as they do not contribute to the enumeration. An optimal pivot pruning strategy would cut the branches that lead to these leaves, leaving only leaves that return a maximal biclique. Let us note l the total number of leaves in the call trees. Some of these leaves return maximal bicliques (counted in $\mathcal{B}$), and others do not. We are then interested in the ratio of "good" leaves:

$$r = \frac{\mathcal{B}}{l}.$$

In particular, if r is less than 1, it means that there are unnecessary recursive calls. Using this ratio, we establish Theorem 4.2 to express the complexity of Algorithm 2 as a function of its output.

Theorem 4.2. *Using the above definition, we have* $1 \leq \frac{1}{r} \leq 2^q$*, and the complexity of Algorithm 2 is in* $\emptyset \frac{1}{r} \cdot (d_U + d_{2U}) \cdot d \cdot q \cdot \mathcal{B}$.

Proof. First, we show that $1 \leq \frac{1}{r} \leq 2^q$. On the one hand, it is clear that $\frac{1}{r} \geq 1$ by definition of r. On the other hand, each maximal biclique of G contains at most 2^q sub-bicliques, so there are at most $2^q \cdot \mathcal{B}$ bicliques in total in the graph. Now, observe that the set R associated to a node of the call trees is a biclique, and the root call to the `BipBronKerbosch` function at iteration i of the loop at Line 1 enumerates all bicliques R that contain x_i and none of the vertices $x_1, \ldots, x_{i-1}$. Consequently, the root call of each iteration enumerates a set of bicliques R that is disjoint from the sets of bicliques resulting from other iterations. The same applies to the recursive calls made in the loop at Line 17: as each vertex processed in an iteration is placed in X, no biclique R in subsequent iterations can contain that vertex. So, each node in the call trees of `BipBronKerbosch` is associated to a biclique different from any other node in any other call tree, so that there are at most $2^q \cdot \mathcal{B}$ nodes in the trees. Thus, as each leaf is a particular node of a tree, we deduce that $l \leq 2^q \cdot \mathcal{B}$, and therefore $\frac{1}{r} \leq 2^q$.

Now, we express the complexity of the BBK algorithm. By definition of q, we know that the depth of any call tree is at most q, thus there are at most $q \cdot l$ nodes in the call trees. Besides, the number of vertices in the set $P \cup X$ that can augment the current biclique is in $\emptyset d_U + d_{2U}$. The pivot is therefore chosen in $\emptyset(d_U + d_{2U}) \cdot d$, by calculating the size of $P \cap N(p)$ for each $p \in P \cup X$. Furthermore, the intersections of the sets P_U, P_V, X_U and X_V with $N(x)$ within the loop starting at Line 17 are done in $\emptyset d$, and this loop iterates over at most $|P|$ vertices, so it runs in $\emptyset(d_U + d_{2U}) \cdot d$ too.

So, the complexity of Algorithm 2 is in $\emptyset(d_U + d_{2U}) \cdot d \cdot q \cdot l$. As $l = \frac{1}{r} \cdot \mathcal{B}$, this complexity can be expressed as $\emptyset \frac{1}{r} \cdot (d_U + d_{2U}) \cdot d \cdot q \cdot \mathcal{B}$. □

This expression of the complexity as a function of output characteristics gives an insight on how close BBK is from an optimal enumeration. Indeed, to enumerate the maximal bicliques, we need at least to write each of them into the output, which is achieved in $\emptyset q \cdot \mathcal{B}$, our algorithm is therefore a factor $\frac{1}{r} \cdot (d_U + d_{2U}) \cdot d$ away from this value. We will evaluate in Section 5 typical values of the factor $\frac{1}{r}$ on real data and see that it is close to 1 in general.

Note that Chen *et al.* also give an expression of the complexity of OOMBEA as a function of its output [10]: it runs in $\emptyset \zeta_U \cdot m \cdot \mathcal{B}$, so comparing these two complexities leads to comparing the factors $\frac{1}{r} \cdot (d_U + d_{2U}) \cdot d \cdot q$ to $\zeta_U \cdot m$, which is not trivial, so we perform an extensive experimental comparison of the running time of both algorithms in the next section.

5 Experiments

In this section, we perform experiments on the BBK algorithm to demonstrate its practical efficiency. We have implemented this algorithm in `C++`, and the code is available online [2]. Throughout this section, the bipartite graphs $G = (U, V, E)$ used are such that the set U is the one

[2] `https://gitlab.lip6.fr/baudin/bbk`

containing the *fewest* vertices and the set V is the one containing the *most*. Unless specified otherwise, we initialize the algorithm on the set U.

We first present the bipartite graphs that are used in these experiments, and compare the execution times of BBK algorithm on these graphs with those of OOMBEA algorithm [10]. Then, we discuss the influence of the choice of the set on which we initialize the enumeration on the vertices (Line 1 of Algorithm 2). Finally, we show that although our implementation leads to shorter execution times, the OOMBEA algorithm is usually more economical in terms of memory consumption.

5.1 Datasets

We tested the BBK algorithm on a set of bipartite graphs retrieved from the KONECT [22] database. We selected bipartite graphs from different real-life situations, corresponding to various numbers of vertices and edges, in order to test the algorithm in different scenarios.

Type of data. The bipartite graphs that we use are presented in Table 1, together with some of their relevant properties. Some of them concern users actions on online platforms: tags posted in *BibSonomy* and *CiteULike*, books rated on *BookCrossing*, movies rated on *MovieLens*, *DVD-Ciao*, *FilmTrust* and *WikiLens*, posts made on forums in *UC-Forum*. Other graphs link people to their activity: *Actor-Movie* is a graph linking actors to the movies that they have starred in, *CiteSeer* links scientific authors to their publications, *GitHub* links users to the projects they are working on. Finally, the remaining graphs correspond to various types of information classification: *DailyKos*, *Reuters* and *NIPS-Papers* connect documents and the words that they contain, *DBpedia* associates athletes to their teams, *TV-Tropes* links artistic works to their style, *Discogs* links musical content to its style, *Marvel* links Marvel comics characters to the publications in which they appear, *Pics* connects people to the images on which they are tagged, and *YouTube* connects users to the groups to which they belong.

Graphs in Table 1 are sorted by increasing number of maximal bicliques (column $\boldsymbol{\mathcal{B}}$). As mentioned above, the two sets of vertices U and V of these bipartite graphs $G = (U, V, E)$ are chosen in such a way that U contains the fewest vertices ($n_U \leq n_V$). *DVD-Ciao* is the graph containing the most bicliques that could be enumerated within a week of computation, while no algorithm was able to terminate for *NIPS-Papers* and *MovieLens* within this computation time limit. It is worth noticing that there is no simple relation between the number of maximal bicliques and the number of edges or vertices. For example, the bipartite graph *FilmTrust* has numbers of edges and vertices of the same order of magnitude as the graph *WikiLens*, but it contains more than 200 times more maximal bicliques.

Bidegeneracies of graphs and vertices. Table 2 presents the maximum degrees $\boldsymbol{d_U}$ and $\boldsymbol{d_V}$ in the bipartite graphs of Table 1, as well as their bidegeneracies $\boldsymbol{b_U}$ and $\boldsymbol{b_V}$ defined in Section 3. We also report the maximum degrees of the graph projected onto U or V ($\boldsymbol{d_{2U}}$ or $\boldsymbol{d_{2V}}$), and the average bidegeneracies $\overline{\boldsymbol{b_U}}$ and $\overline{\boldsymbol{b_V}}$ that are respectively the mean of the bidegeneracies of the vertices of U and of V. We remind that the complexities of the BBK algorithm have been expressed in Theorem 4.1 with b_U (or similarly b_V), thanks to the nodes processed in a bidegeneracy order. With a random order, this factor would be bounded by $d_{2U} + d_U$ (or $d_{2V} + d_V$). We observe that, while b_U (and b_V) $\geq max(d_U, d_V)$, we almost always have $b_U = max(d_U, d_V)$. Finally, b_U (and b_V) are lower than $d_{2U} + d_U$ (or $d_{2V} + d_V$) but typically of the same order of magnitude. So, the bidegeneracies of the graphs, which appear in the complexity expressions, do not give a clear understanding of the computational gain that the bidegeneracy order brings. Moreover, when considering the average values of the vertex bidegeneracies ($\overline{\boldsymbol{b_U}}$ and $\overline{\boldsymbol{b_V}}$), we can see that the

Graph	m	n_U	n_V	$\mathcal{B}$	Source: http://konect.cc/networks/
UC-Forum	7,089	522	899	16,261	opsahl-ucforum
Discogs	481,661	15	270,771	17,650	discogs_lgenre
CiteSeer	512,267	105,353	181,395	171,354	komarix-citeseer
Marvel	96,662	6,486	12,942	206,135	marvel
DBpedia	1,366,466	34,461	901,130	517,943	dbpedia-team
Actor-Movie	1,470,404	127,823	383,640	1,075,444	actor-movie
Pics	997,840	17,122	495,402	1,242,718	pics_ui
YouTube	293,360	30,087	94,238	1,826,587	youtube-groupmemberships
WikiLens	26,937	326	5,111	2,769,773	wikilens-ratings
BookCrossing	1,149,739	105,278	340,523	54,458,953	bookcrossing_full-rating
GitHub	440,237	56,519	120,867	55,346,398	github
DailyKos	353,160	3,430	6,906	242,384,960	bag-kos
FilmTrust	35,494	1,508	2,071	646,318,495	librec-filmtrust-ratings
CiteULike	538,761	22,715	153,277	2,333,281,521	citeulike-ut
Reuters	978,446	19,757	38,677	10,071,287,092	gottron-reuters
BibSonomy	453,987	5,794	204,673	10,526,275,315	bibsonomy-2ut
TV-Tropes	3,232,134	64,415	87,678	19,636,996,096	dbtropes-feature
DVD-Ciao	1,625,480	21,019	71,633	109,769,732,096	librec-ciaodvd-review_ratings
NIPS-Papers	746,316	1,500	12,375	-	bag-nips
MovieLens	1,000,009	3,760	6,040	-	movielens-1m

Table 1: Datasets used in the experiments, sorted by increasing number of maximal bicliques. m is the number of links in the bipartite graph, n_U the number of vertices in set U, n_V the number of vertices in its set V (denominated such that $n_U \leq n_V$), and $\mathcal{B}$ the number of maximal bicliques. A "-" symbol means that we do not know the number of maximal bicliques, as no algorithm finishes in less than a week for these graphs.

average vertex bidegeneracy is often significantly lower than the bidegeneracy of the graph. As noticed in Section 4.1, the complexity as a function of the input can be more precisely expressed as $\emptyset \sum_{i=1}^{n_U} \left((b(u_i) + |X_i|) \cdot 3^{b(u_i)/3} \right)$, where the index i is given by a bidegeneracy order and X_i is defined at Line 3 of Algorithm 2. So the fact that vertex bidegeneracies are relatively lower gives a clue as to how well the algorithm works in practice on these graphs: the worst cases of bidegeneracy are not called up on many vertices.

5.2 Results: computation time

To evaluate the gain in efficiency in the enumeration of maximal bicliques, we measure the computation times of our implementation and compare them to the ones obtained with the Chen *et al.* [10] implementation[3]. We carry out these experiments on machines equipped with 2 Intel Xeon E5645 processors with 12 cores each at 2.4 GHz and 128 GB of RAM. We set a computation time limit of one week, so that computations that exceed this limit are interrupted.

Figure 2 presents these computation times on a logarithmic scale and the corresponding numerical values are detailed in Table 3 (values "-" in the table correspond to computations that did not finish within one week). Graphs are sorted by increasing number of maximal bicliques. We can observe that the computation time generally increases with the number of bicliques, whereas it does not seem to be directly related to the number of vertices, edges, degree or bidegeneracy. In most cases, the computation time is lower for BBK than that for OOMBEA, and the more bicliques the graph contains, the larger the difference, reaching more than a factor

[3] https://github.com/S1mpleCod/cohesive_subgraph_bipartite

Graph	d_U	d_V	d_{2U}	d_{2V}	b_U	b_V	$\overline{b_U}$	$\overline{b_V}$
UC-Forum	126	99	411	634	126	144	91	95
Discogs	128,070	15	15	270,771	128,070	128,070	32,110	102,088
CiteSeer	286	385	596	1,653	385	385	9.3	43
Marvel	1,625	111	1,934	9,855	1,625	1,625	40	769
DBpedia	2,671	17	2,839	18,517	2,671	2,671	47	522
Actor-Movie	294	646	7,799	3,957	646	646	171	47
Pics	7,810	335	7,079	113,079	7,810	7,810	173	1,459
YouTube	7,591	1,035	7,357	37,514	7,591	7,591	124	1,049
WikiLens	1,721	80	285	4,826	1,721	1,721	138	1,062
BookCrossing	13,601	2,502	53,916	151,646	13,601	13,601	215	1,669
GitHub	884	3,675	15,995	29,650	3,675	3,675	525	103
DailyKos	457	2,123	430	6,895	2,817	2,123	2,808	1,349
FilmTrust	244	1,044	1,459	1,770	1,100	1,044	1,020	152
CiteULike	4,072	8,814	18,190	80,410	8,814	8,814	3,903	915
Reuters	380	19,044	19,731	37,716	19,044	19,044	18,632	287
BibSonomy	21,463	1,407	4,614	159,465	21,463	21,463	584	6,919
TV-Tropes	6,507	12,400	47,460	37,494	12,400	12,400	6,487	2,331
DVD-Ciao	34,884	422	13,000	62,027	34,884	34,884	241	24,195
NIPS-Papers	914	1,455	1,500	12,363	1,760	3,312	1,755	2,671
MovieLens	3,428	2,314	3,660	6,040	3,429	4,998	2,492	4,978

Table 2: Characteristics of the bipartite graphs of Table 1. d_U and d_V are the maximum degrees in U and V, d_{2U} and d_{2V} are the maximum projected degrees on U and V, b_U and b_V are the maximum bidegeneracies and $\overline{b_U}$ and $\overline{b_V}$ are the mean bidegeneracies of U and V.

10 for *FilmTrust* and *CiteULike*. Finally, for the graphs with the largest number of maximal bicliques, OOMBEA does not obtain all the maximal bicliques within the time limit of one week of computing.

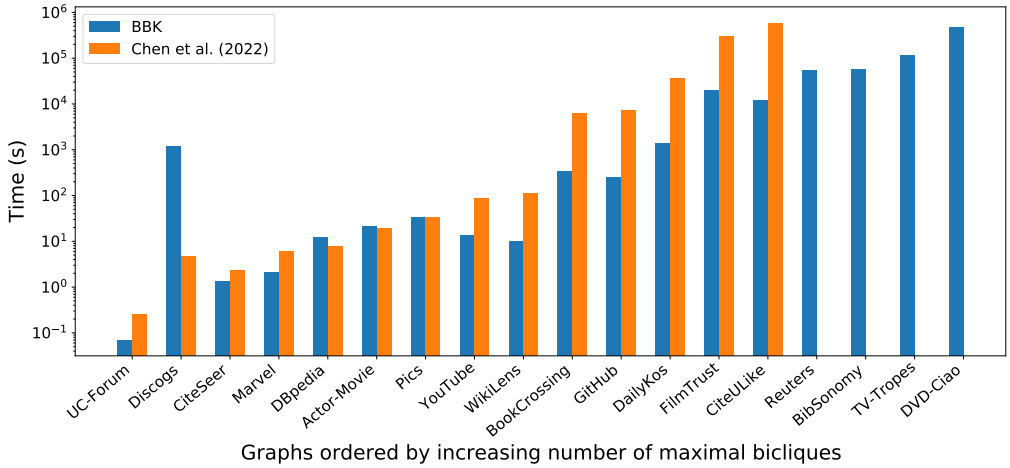

Figure 2: Computation times of the BBK algorithm on the datasets of Table 1 compared to those of the OOMBEA algorithm. On the rightmost graphs, the values for OOMBEA are not displayed because the computation was not completed within the one week limit of the experiments.

However, there are some graphs (among those with the least maximal bicliques) for which OOMBEA is faster than BBK: this is the case for *Discogs*, *DBpedia* and *Actor-Movie*. *Discogs* is a unique case in our experiments, for which OOMBEA is much faster than our algorithm. It stems from the fact that the structure of this graph is very particular as its set U is small ($n_U = 15$), but it has a high average bidegeneracy ($\overline{b_U} > 10^4$), while BBK tends to be more efficient on graphs with lower bidegeneracy. By contrast, the *unilateral coreness* value ζ_U present in Chen *et al.* complexity expression is bounded by the size n_U of U, as it is based on the projection onto U, which makes OOMBEA significantly more efficient on this instance.

We also display in Table 3 the ratio r defined in Section 4.2. We remind that this ratio corresponds to the fraction of leaves in the call tree that return a maximal biclique; the other leaves are unnecessary for the computation, and would be pruned by an optimal pivot strategy. We can see that this ratio is relatively close to 1 in our enumerations, which means that a better pivot could not be much more efficient at pruning useless branches of the call trees. Note that three graphs stand out with a lower r values: *DBpedia*, *Actor-Movie* and *Pics*. Interestingly, these are the cases where BBK performs relatively poorly by comparison with OOMBEA in terms of computation time (excluding *Discogs* discussed above). This suggests that the reason for this poorer performance is a lower efficiency of the pivot pruning on these graphs.

5.3 Starting the enumeration from U or V

In Algorithm 2, we can choose to run the loop on Line 1 on the vertices of U or on the vertices of V. While the output of the algorithm is identical, we show here that this choice may have a significant impact on the computation time.

By default, the iterations are carried out on the set U containing the lowest number of vertices, and we use this case as a reference. Figure 3 shows the impact of iterating on the set containing the most vertices, which is the set V in Table 1. The bars represented in this figure correspond

Graph	t_{BBK}	t_{OOMBEA}	r
UC-Forum	0.07	0.26	0.60
Discogs	1,185	4.6	1.00
CiteSeer	1.4	2.3	0.71
Marvel	2.1	6.1	0.66
DBpedia	12	7.7	0.40
Actor-Movie	21	19	0.21
Pics	32	34	0.39
YouTube	13	87	0.76
WikiLens	10	111	0.91
BookCrossing	335	6,169	0.77
GitHub	248	7,283	0.84
DailyKos	1,419	35,837	0.77
FilmTrust	20,255	300,307	0.98
CiteULike	12,338	594,549	0.92
Reuters	54,045	-	0.82
BibSonomy	58,719	-	0.96
TV-Tropes	113,659	-	0.73
DVD-Ciao	476,686	-	0.93
NIPS-Papers	-	-	-
MovieLens	-	-	-

Table 3: Computation times (in seconds) obtained by BBK and OOMBEA algorithms. A "-" symbol means that the computation has not been completed within one week. The last column represents the ratio r for BBK defined in Section 4.2 and which appears in the expression of the complexity (Theorem 4.2).

to the time of a run on V divided by the time by a run on U, so a bar above $y = 1$ means that the run is slower when iterating on the larger set of vertices, and vice versa.

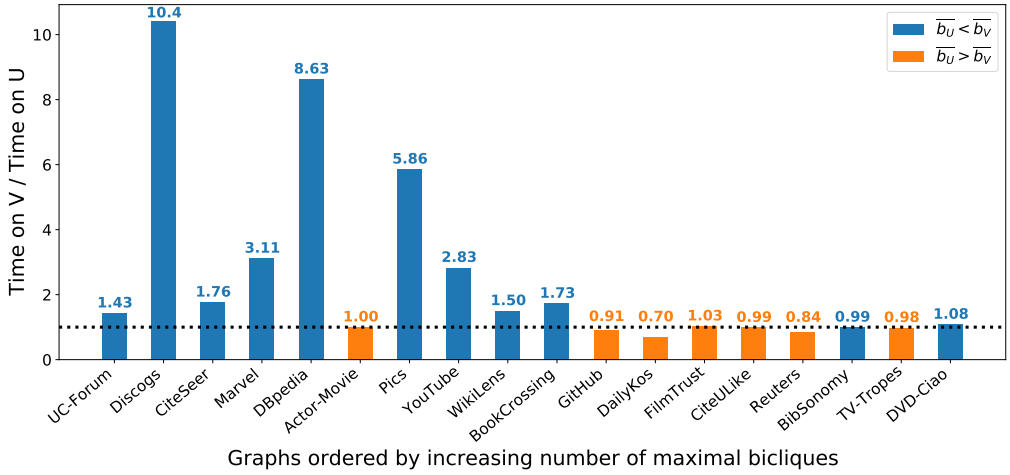

Figure 3: Ratio between the execution time of Algorithm 2 BBK when the run is performed on the larger set V to the execution time when the run is performed on the smaller set U. The results in blue correspond to graphs where $\overline{b_U} < \overline{b_V}$, and in orange to graphs where $\overline{b_U} > \overline{b_V}$.

First, we notice that this choice can have a strong impact on the computation time, since for some graphs such as *Discogs*, *DBpedia* or *Pics*, the computation time varies by a factor larger than 5, while it seems to be less significant for the graphs containing the most maximal bicliques, that have a factor closer to 1. Then, we observe that choosing the smallest set U is appropriate for two-thirds of the datasets, but there are several graphs where it is more efficient to perform the run on V. We have seen that if a vertex u has a bidegeneracy $b(u)$, then the time spent in the call of function `BipBronKerbosch` made at the iteration of the loop of Line 1 corresponding to this vertex is exponential in $b(u)$ (see the proof of Theorem 4.1). Consequently, we investigate if there is a relation between the computation times in regard to $\overline{b_U}$ and $\overline{b_V}$ to understand the origin of these observations.

Thus, we distinguish between two cases: we color a bar in Figure 3 in orange when $\overline{b_U} > \overline{b_V}$ and in blue when $\overline{b_V} > \overline{b_U}$. It appears that, almost in all cases, initializing on the set with the lowest mean bidegeneracy is the most efficient choice; the exceptions are *FilmTrust* and *BibSonomy*, for which the choice between U or V has little impact on the computation time.

5.4 About the memory usage

While BBK algorithm is in general more efficient than OOMBEA in terms of computation time, the OOMBEA implementation proposed by Chen *et al.* [10] is more economical when considering memory usage. To illustrate this, Figure 4 shows the memory used by BBK and OOMBEA in logarithmic scale on the datasets of Table 1.

Figure 4 shows that OOMBEA is able to enumerate the maximal bicliques using typically ten times less memory than BBK (even more for the *Discogs* special case). One explanation for this is that to gain efficiency within a recursive call, we use the method described by Eppstein *et al.* [16]: it consists in pre-allocating for each vertex a list of the size of its bidegeneracy. As a vertex cannot belong to a biclique containing more vertices than its bidegeneracy, this size is a bound on the depth of the tree of calls to `BipBronKerbosch` related to this vertex. These lists

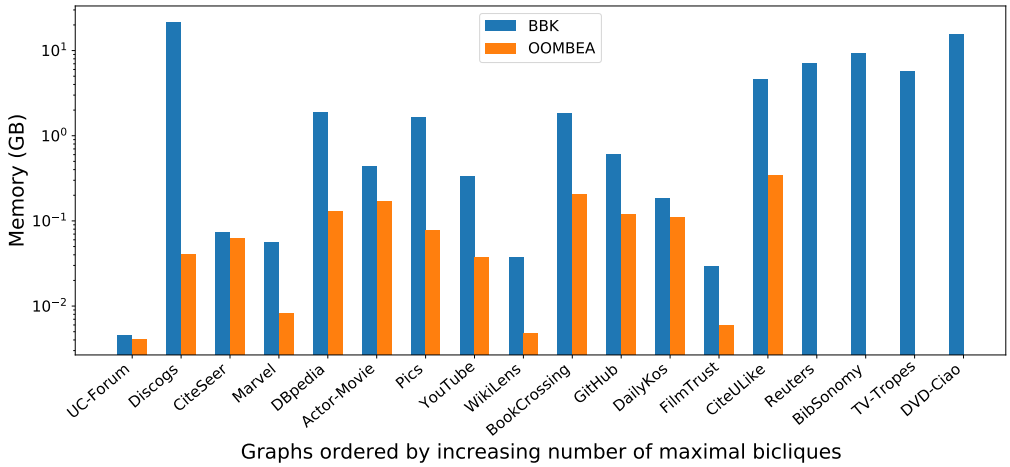

Figure 4: Memory used by the two algorithms BBK and OOMBEA on the datasets of Table 1. For the four rightmost graphs, OOMBEA cannot complete the enumeration in less than one week, so its result is not displayed.

are then used to record the end index of each vertex adjacency list used on a given recursive call: it allows reducing the size of the used neighborhoods while avoiding wasting time copying them, although it adds a factor $n_U \cdot \overline{b_U}$ to the memory complexity of the algorithm. In bipartite graphs, the average bidegeneracy of vertices can be relatively large (see Table 2), which implies that storing this list may result in high memory requirements.

Note however that except for five graphs, the memory used by BBK does not exceed 2 GB, and never exceeds about 20 GB, which is largely manageable on most modern computers. So, in practice, memory is not the limiting factor for the enumeration of maximal bicliques, which is why we favor an algorithm aiming at time-efficiency.

6 Conclusion

In this article, we have introduced the BBK algorithm for enumerating maximal bicliques in bipartite graphs, which aims at being time-efficient. To do this, we have adapted the recursive Bron-Kerbosch algorithm to the context of bipartite graphs by using an extended graph on which finding cliques is equivalent to finding bicliques in the original instance. Moreover, we take advantage of the sparsity of real-world graphs by formulating the algorithm so as to use the neighborhood in the original bipartite graph only. To improve its efficiency, the algorithm processes the vertices of the graph in an order that we call bidegeneracy order, which aims at reducing the set of candidate vertices at each recursive call. We also add to the process a classic pivot-based pruning strategy, adapted to the context of bipartite graphs. We have carried out a theoretical analysis of the BBK algorithm to establish two complexity expressions: one as a function of its input and one as a function of its output characteristics. Finally, we provide an open-source C++ implementation of BBK, which we have used to illustrate the good performances experimentally on massive real-world datasets. These experiments shown that BBK can enumerate maximal bicliques typically 10 times faster than the state of the art does on larger instances, and produces results in cases where the state of the art is unable to provide a solution within one week of computation.

We identify several directions which can be developed from this work. One of them is the search for bicliques in non-bipartite graphs. This question has been explored extensively probably because bicliques play an important role in the structure of real-world graphs such as protein interaction networks [25]. Indeed, as our approach essentially uses the neighbors and second neighbors of a node, it should be translatable to this context, yet concepts such as the vertex bidegeneracy order would have to be adapted accordingly. Another interesting lead comes from the fact that finding bicliques in large bipartite graphs is similar to detecting closed itemsets in transaction databases, as mentioned earlier [26, 34]. Precisely, if we map items to set U and transactions to V, an itemset with support larger than s would be a subset of U that forms a biclique with at least s vertices of V. Thus, adapting BBK to this specific issue may bring new, efficient solutions to this problem.

Acknowledgments

This work is funded in part by the ANR (French National Agency of Research) through the FiT-ANR-19-LCV1-0005 grant.

References

[1] A. Abidi, R. Zhou, L. Chen, and C. Liu. Pivot-based maximal biclique enumeration. In *IJCAI*, pages 3558–3564, 2020.

[2] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Santiago, Chile, 1994.

[3] V. Batagelj and M. Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[4] A. Baudin, C. Magnien, and L. Tabourier. Faster maximal clique enumeration in large real-world link streams. *arXiv preprint arXiv:2302.00360*, 2023.

[5] C. Borgelt. Frequent item set mining. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 2(6):437–456, 2012.

[6] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1-6):309–320, 2000.

[7] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[8] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, et al. Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic acids research*, 31(9):2443–2450, 2003.

[9] S.-C. Chabert-Liddell, P. Barbillon, and S. Donnet. Impact of the mesoscale structure of a bipartite ecological interaction network on its robustness through a probabilistic modeling. *Environmetrics*, 33(2):e2709, 2022.

[10] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li. Efficient maximal biclique enumeration for large sparse bipartite graphs. *Proceedings of the VLDB Endowment*, 15(8):1559–1571, 2022.

[11] D. Coates, I. Naidenova, and P. Parshakov. Transfer policy and football club performance: evidence from network analysis. *International Journal of Sport Finance*, 15(3):95–109, 2020.

[12] A. Conte and E. Tomita. On the overall and delay complexity of the cliques and bron-kerbosch algorithms. *Theoretical Computer Science*, 899:1–24, 2022.

[13] P. Damaschke. Enumerating maximal bicliques in bipartite graphs with favorable degree sequences. *Information Processing Letters*, 114(6):317–321, 2014.

[14] A. Das, S.-V. Sanei-Mehri, and S. Tirthapura. Shared-memory parallel maximal clique enumeration. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 62–71. IEEE, 2018.

[15] C. F. Dormann, J. Fründ, N. Blüthgen, and B. Gruber. Indices, graphs and null models: analyzing bipartite ecological networks. 2009.

[16] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.

[17] A. Gély, L. Nourine, and B. Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete applied mathematics*, 157(7):1447–1459, 2009.

[18] J.-L. Guillaume and M. Latapy. Bipartite graphs as models of complex networks. *Physica A: Statistical Mechanics and its Applications*, 371(2):795–813, 2006.

[19] D. Hermelin and G. Manoussakis. Efficient enumeration of maximal induced bicliques. *Discrete Applied Mathematics*, 303:253–261, 2021.

[20] R. F. Hriez, G. Al-Naymat, and A. Awajan. An effective algorithm for extracting maximal bipartite cliques. In *International Conference on Data Science, E-learning and Information Systems 2021*, pages 76–81, 2021.

[21] W. Huber, V. J. Carey, L. Long, S. Falcon, and R. Gentleman. Graphs in molecular biology. *BMC bioinformatics*, 8(6):1–14, 2007.

[22] J. Kunegis. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pages 1343–1350, 2013. Données disponibles à l'adresse: `http://konect.cc/networks`.

[23] S. Lehmann, M. Schwartz, and L. K. Hansen. Biclique communities. *Physical review E*, 78(1):016108, 2008.

[24] Y. Li, A. Wen, Q. Lin, R. Li, and Z. Lu. Name disambiguation in scientific cooperation network by exploiting user feedback. *Artificial Intelligence Review*, 41:563–578, 2014.

[25] H.-B. Liu, J. Liu, and L. Wang. Searching maximum quasi-bicliques from protein-protein interaction network. *Journal of Biomedical Science and Engineering*, 1(3):200, 2008.

[26] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Algorithm Theory-SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory, Humlebæk, Denmark, July 8-10, 2004. Proceedings 9*, pages 260–272. Springer, 2004.

[27] A. S. Muhammad, P. Damaschke, and O. Mogren. Summarizing online user reviews using bicliques. In *SOFSEM 2016: Theory and Practice of Computer Science: 42nd International Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 23-28, 2016, Proceedings 42*, pages 569–579. Springer, 2016.

[28] M. E. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review E*, 64(1):016132, 2001.

[29] G. A. Pavlopoulos, P. I. Kontou, A. Pavlopoulou, C. Bouyioukos, E. Markou, and P. G. Bagos. Bipartite graphs in systems biology and medicine: a survey of methods and applications. *GigaScience*, 7(4):giy014, 2018.

[30] C. Qin, M. Liao, Y. Liang, and C. Zheng. Efficient algorithm for maximal biclique enumeration on bipartite graphs. In *Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery: Volume 2*, pages 3–13. Springer, 2020.

[31] B. I. Simmons, A. R. Cirtwill, N. J. Baker, H. S. Wauchope, L. V. Dicks, D. B. Stouffer, and W. J. Sutherland. Motifs in bipartite ecological networks: uncovering indirect interactions. *Oikos*, 128(2):154–170, 2019.

[32] O. Voggenreiter, S. Bleuler, and W. Gruissem. Exact biclustering algorithm for the analysis of large gene expression data sets. *BMC bioinformatics*, 13(Suppl 18):A10, 2012.

[33] S. Yoon, L. Benini, and G. De Micheli. Co-clustering: a versatile tool for data analysis in biomedical informatics. *IEEE Transactions on Information Technology in Biomedicine*, 11(4):493–494, 2007.

[34] M. J. Zaki and M. Ogihara. Theoretical foundations of association rules. In *3rd ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, pages 71–78, 1998.

[35] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics*, 15:1–18, 2014.

[36] T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang. Bipartite network projection and personal recommendation. *Physical review E*, 76(4):046115, 2007.

[37] Z. Zhu, J. Su, and L. Kong. Measuring influence in online social network based on the user-content bipartite graph. *Computers in Human Behavior*, 52:184–189, 2015.