Towards Effective Exact Methods for the Maximum Balanced Biclique Problem in Bipartite Graphs

Yi Zhou [a,b], André Rossi [b], Jin-Kao Hao [b,c,*]

[a] *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*

[b] *LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers, France*

[c] *Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France*

European Journal of Operational Research, 2018
https://doi.org/10.1016/j.ejor.2018.03.010

Abstract

The Maximum Balanced Biclique Problem (MBBP) is a prominent model with numerous applications. Yet, the problem is NP-hard and thus computationally challenging. We propose novel ideas for designing effective exact algorithms for MBBP in bipartite graphs. First, an Upper Bound Propagation (UBP) procedure to precompute an upper bound involving each vertex is introduced. Then we extend a simple Branch-and-Bound (B&B) algorithm by integrating the pre-computed upper bounds. Based on UBP, we also study a new integer linear programming model of MBBP which is more compact than an existing formulation [5]. We introduce new valid inequalities induced from the upper bounds to tighten these mathematical formulations for MBBP. Experiments with random bipartite graphs demonstrate the efficiency of the extended B&B algorithm and the valid inequalities generated on demand. Further tests with 30 real-life instances show that, for at least three very large graphs, the new approaches improve the computational time with four orders of magnitude compared to the original B&B.

Keywords: Combinatorial optimization; Clique; Exact algorithms; Techniques for tight bounds; Mathematical formulation.

* Corresponding author.
Email address: jin-kao.hao@univ-angers.fr (Jin-Kao Hao).

1 Introduction

Given a bipartite graph $G = (U, V, E)$ with two disjoint vertex sets U, V and an edge set $E \subseteq U \times V$, a biclique $A \cup B$ (or (A, B)) is the union of two subsets of vertices $A \subseteq U$, $B \subseteq V$ such that $\forall i \in A, \forall j \in B, \{i, j\} \in E$. In other words, the subgraph induced by vertex set $A \cup B$ is a complete bipartite graph. If $|A| = |B|$, then biclique (A, B) is a balanced biclique. The Maximum Balanced Biclique Problem (MBBP) is to find a balanced biclique (A, B) of maximum cardinality. As $|A| = |B|$ holds for a balanced biclique (A, B), MBBP is then to find the maximum half-size balanced biclique. MBBP is a special case of the conventional maximum clique problem [17].

MBBP is a prominent model with a large range of applications, such as nanoelectronic system design [1,14], biclustering of gene expression data in computational biology [4] and PLA-folding in the VLSI theory [10]. In all these applications, the given graphs are bipartite graphs. In terms of computational complexity, the decision version of MBBP is NP-Complete [6,2], though the maximum biclique problem in bipartite graphs (without requiring $|A| = |B|$) is polynomially solvable by the maximum matching algorithm [4].

Considerable effort has been devoted to the pursuit of effective algorithms for MBBP in bipartite graphs, both theoretically and practically. Heuristic algorithms represent the most popular approach for MBBP, though they do not guarantee the optimality of the attained solutions. The majority of existing heuristic algorithms solve the equivalent maximum balanced independent set (a vertex set such that no two vertices are adjacent) problem in the complement bipartite graph, rather than directly seeking the maximum balanced biclique from the given graph. For example, several greedy heuristic algorithms were proposed, which apply vertex-deletion rules on the complement bipartite graph in the period from 2006 to 2014 [1,14,18,19], while an evolutionary algorithm combining structure mutation and repair-assisted restart was introduced in 2015 [20].

On the other hand, according to our literature review, there are only two studies on exact algorithms. In [14], a recursive exact algorithm for searching a maximum balanced independent set with a given half-size in the complement graph was proposed. However, the computational time of this algorithm becomes prohibitive when the number of vertices of the given graph exceeds (32,32). In [9], a Branch-and-Bound (B&B) algorithm for MBBP (named BB-Clq) for general graphs (including non-bipartite graphs) was studied. The algorithm incorporates a clique cover technique for upper bound estimation (an equivalent technique of using graph coloring to estimate the upper bound for the maximum clique problem) and employs lex symmetry breaking techniques for general graphs. As far as we know, this algorithm is currently the

best performing exact algorithm, even though the bounding technique and symmetry breaking techniques are only effective for non-bipartite graphs.

In addition to specifically designed exact algorithms, the general Integer Linear Programming (ILP) constitutes an interesting alternative for solving hard combinatorial problems such as MBBP. Commercial mixed ILP solvers, like IBM CPLEX, can even solve some hard instances which cannot be handled by other approaches. Meanwhile, the success of a ILP solver highly depends on the tightness of the mathematical formulation of the problem. For MBBP, an ILP formulation has been proposed in [5], which is based on the complement graph. Another mathematical formulation that defines the constraints on the original graph was presented in [20]. However, this formulation was not applicable for ILP solvers as it contains non-linear constraints.

In this work, we introduce new ideas for developing effective exact algorithms for MBBP, which help to solve very large MBBP instances from applications like social networks. Our main contributions can be summarized as follows.

First, we elaborate an Upper Bound Propagation (UBP) procedure inspired from [13], which produces an upper bound of the maximum balanced biclique involving each vertex in the bipartite graph. UBP propagates the initial upper bound involving each vertex and achieves an even tighter upper bound for each vertex. UBP is independent from the search procedure and is performed before the start of the search algorithm. An extended exact algorithm, denoted by ExtBBClq, is proposed by taking advantage of UBP to improve BBClq, the branch-and-bound algorithm introduced in [9].

Second, we introduce a new and more compact formulation that requires a largely reduced number of constraints compared to the previous formulation presented in [5]. In the previous model of [5], the number of constraints equals the number of edges in the complement bipartite graph, making it inapplicable to solve large real-life sparse graphs. The proposed model reduces the number of constraints to the number of vertices in the graph, which allows for dealing with very large instances. We also introduce new inequalities to tighten both previous and new formulations. Our computational results suggest that the new formulation and tightened inequalities improve the performance of the ILP solver CPLEX.

The remainder of the paper is organized as follows. Section 2 introduces the notations that will be used throughout the paper and Section 3 reviews the BBClq algorithm introduced in [9]. In Section 4, we present our Upper Bound Propagation procedure for upper bound estimation and explain how to use it to improve BBClq. In Section 5, we discuss the existing ILP formulation for MBBP and present our new ILP model. We also study how the upper bounds can lead to new valid inequalities to tighten the ILP formulations.

Computational results and experimental analyses are presented in Section 6, followed by conclusions and future working directions.

2 Notations

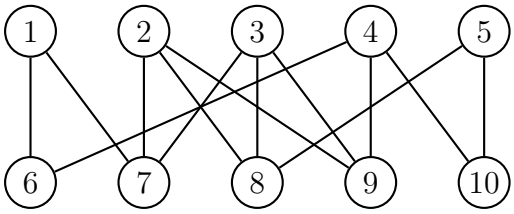

Fig. 1. A bipartite graph $G = (U, V, E)$, $U = \{1, 2, 3, 4, 5\}$, $V = \{6, 7, 8, 9, 10\}$.

Given a bipartie graph $G = (U, V, E)$ ($|U| \leq |V|$ if not specifically stated), let $(A, B) \subseteq (U, V)$ be a balanced biclique of G (i.e., $|A| = |B|$). The half-size of the balanced biclique (A, B) is the cardinality of $|A|$ (or $|B|$). For example, in Figure 1, $(\{2, 3\}, \{7, 8\})$ is a balanced biclique of half-size of 2. For all $S \subseteq U \cup V$, $G[S]$ denotes the subgraph of G induced by S. Given a vertex i in G, the set of vertices adjacent to i is denoted by $N(i) = \{j : \{i, j\} \in E\}$ and $deg_G(i) = |N(i)|$ is the degree of vertex i. The upper bound involving vertex i, denoted by ub_i, is an upper bound of the half-size of the maximum balanced biclique containing vertex i. For example, in Figure 1, a possible value for ub_1 could be 2, since $deg_G(1) = 2$.

3 Review of the BBClq algorithm

To our knowledge, the B&B algorithm BBClq presented in [9] is the current best-performing exact algorithm for MBBP for general graphs. The algorithm is mainly inspired from the well-known algorithms [15,12] for the maximum clique problem. Algorithm 1 shows the general search scheme of BBClq.

BBClq recursively builds two sets A and B such that (A, B) forms a biclique. It maintains a candidate set C_A (C_B) that includes vertices which are eligible to join A (B) while ensuring that (A, B) is a biclique (i.e., $C_A = \bigcap_{i \in B} N(i)$, $C_B = \bigcap_{i \in A} N(i)$). Initially, the algorithm sets lb, the global lower bound on the maximum biclique half-size to 0 and starts the search by calling BBClq$(G, \emptyset, \emptyset, U, V)$.

At each recursive call to BBClq, a vertex i (called branch vertex) is moved from C_A (lines 7-8). The algorithm then considers the branches (possibilities) of $i \in A$ (lines 9-12) and $i \notin A$ in the next `while` loop. The bounding procedure

(line 9) prunes the branch of $i \in A$ if the upper bound after estimation in this context is not larger than the global lower bound. The upper bound estimating method, which is classically a key point concerning the performance of a B&B algorithm, will be introduced in the following section. If the current branch is not pruned, the search goes on by reconstructing A' with a new vertex v and C'_B by filtering from C_B those vertices not adjacent to v (every vertex in B must be adjacent to every vertex in A). After updating the two sets, the algorithm recursively calls BBClq in line 12, swapping the roles of A and B, as A and B are extended alternatively for the sake of satisfying the balance requirement. The above process is repeated in the next recursive call of BBClq.

When the algorithm loops back to line 4, as we just mentioned, it explores another branch implying $i \notin A$. The `while` loop stops when C_A becomes empty or when the remaining vertices in C_A do not allow to build a solution better than the global lower bound (lines 5-6). Besides, since $|A| + 1 = |B|$ or $|A| = |B|$ holds each time BBClq is called, we update the lower bound in lines 1-3 once $|A| > lb$ and store the incumbent solution (A, B) as the best solution found so far. As a result, the best solution (A^*, B^*) is an optimal biclique with $|A^*| = lb$ ($A^* \subseteq U$ or $A^* \subseteq V$), but it may not be totally balanced ($|A^*| - |B^*| \leq 1$). Thus, in line 13, the procedure of retrieving the maximum balanced biclique (of half-size lb) from a biclique is accomplished by *make_balance()*. This procedure simply removes vertices from the larger set A^* or B^* until a balanced biclique is obtained.

Figure 1 is used to illustrate BBClq. Initially, $lb = 0$ and BBClq$(G, \emptyset, \emptyset, U, V)$ is called. According to the minimal degree heuristic in [9], vertex 1 is chosen as the first branch vertex. Clearly, the current upper bound is greater than 0, the algorithm proceeds to BBClq$(G, \emptyset, \{1\}, \{6, 7\}, U \setminus \{1\})$ to explore the solutions containing vertex 1. As a result, the solution $(\{1\}, \{6\})$ is found and lb is updated to 1. Likewise, the algorithm selects 5 as the second branch vertex in the following loop, proceeds to BBClq$(G, \emptyset, \{5\}, \{8, 10\}, U \setminus \{1, 5\})$ if no upper bounding technique is applied. We can see that this recursive call to BBClq has to explore the case of expanding the given biclique by adding vertex 8 or 10. However, with the upper bounding estimating technique proposed in this paper, the call of BBClq$(G, \emptyset, \{5\}, \{8, 10\}, U \setminus \{1, 5\})$ will not even start since the upper bound involving vertex 5 is 1 ($upper\_bound(\{5\}) = lb = 1$). The algorithm finds the optimal solution $(\{1, 2\}, \{4, 5\})$ after the third loop (which explores $A = \{2\}$ and calls BBClq$(G, \emptyset, \{2\}, \{7, 8, 9\}, U \setminus \{1, 5, 2\})$). There will be no additional iteration as $|A| + |C_A| \leq 2$ ($A = \emptyset, C_A = \{3, 4\}$).

Note that the BBClq version we presented is a trimmed version of the original algorithm in [9] since we mainly focus on solving MBBP in bipartite graphs. First, symmetric breaking is ignored as it is not concerned for bipartite graphs. Second, the original algorithm calculates a clique cover (by considering the graph coloring problem on the complement graph) to estimate

the upper bound in a general graph based on the fact that sets A and B are independent sets. However in bipartite graphs, the upper bound found by this technique is trivial as the two vertex sets of the input graph are necessarily independent sets. Hence, we do not keep this technique in BBClq neither.

Algorithm 1: BBClq(G, A, B, C_A, C_B), the trimmed B&B algorithm for MBBP from [9]

Input: Graph instance $G = (U, V, E)$, A, B - current sets that form a biclique, C_A, C_B - the sets of eligible vertices that can be added to A and B respectively

Output: A maximum balanced biclique of G.

```
1  if |A| > lb then
2      lb ← |A|
3      Record current best biclique in (A*, B*)
4  while C_A ≠ ∅ do
5      if |A| + |C_A| ≤ lb then
6          return
7      i ← branch_vertex(C_A)
8      C_A ← C_A \ {i}
9      if upper_bound(A ∪ {i}) > lb then
10         A' ← A ∪ {i}
11         C'_B ← C_B ∩ N(i)
12         BBClq(G,B, A', C'_B, C_A)
13 return make_balance(A*, B*)
```

4 Upper bound propagation and its use to improve BBClq

In this section, we introduce our Upper Bound Propagation procedure (UBP) which is a pre-processing technique to reinforce the BBClq algorithm presented in the last section.

4.1 The upper bound propagation procedure

For each vertex, an upper bound on the half-size of any maximum balanced biclique involving that vertex. UBP is based on the following propositions.

Proposition 1 *For each vertex $i \in U \cup V$, $deg_G(i)$ is an upper bound on the maximum half-size balanced biclique involving i.*

This proposition is obviously true since the half-size of a balanced biclique cannot exceed the degree of any vertex in the biclique.

Before presenting the second proposition, let us define $w_{ij} = |N(i) \cap N(j)|$ for two vertices $i, j \in U \cup V$.

Proposition 2 *Given a vertex $i \in U$, let y_i be the maximum integer such that there exists at least y_i values in $\{w_{ij} : j \in U\}$ greater than or equal to y_i. Then y_i is an upper bound on the maximum half-size balanced biclique involving i.*

Proof: Clearly, in the maximum balanced biclique (A, B) involving $i \in A$, for any vertex $j \in A$ (including i), we have $B \subseteq N(i) \cap N(j)$. Therefore, the maximum possible value y_i such that y_i vertices in U share at least y_i adjacent vertices with i is an upper bound involving i. Note that this proposition also holds given any vertex in V. ■

Proposition 3 *Given a vertex $i \in U \cup V$, let t_i be the largest integer such that there exists t_i vertices in $N(i)$ having upper bounds at least t_i. Then t_i is an upper bound on the maximum half-size balanced biclique involving i.*

Proof: We prove this proposition by contrapositive. Suppose t_i is not an upper bound, then there exists a balanced biclique (A', B') involving $i \in A'$ of half-size t'_i such that $t'_i > t_i$, implying that all the t'_i vertices in B' ($B' \subseteq N(i)$) must have an upper bound of at least t'_i (i.e., $\forall j \in B', ub_j \geq t'_i$), which contradicts the condition that t_i is the maximum integer such that there exists in $N(i)$ at least t_i vertices having $t_i \geq ub_i$. ■

Consider the example of Figure 1, according to Proposition 1, we have $ub_1 = ub_5 = 2$, $ub_2 = ub_3 = ub_4 = 3$. Then, following Proposition 2, ub_1 can be improved (decreased) to 1 since $w_{12} = w_{13} = w_{14} = 1$, $w_{15} = 0$ ($y_1 = 1$). Similarly ub_2, ub_3, ub_4, ub_5 can also be improved to 2, 2, 1, 1 respectively. By Proposition 3, it can be deduced that $ub_6 = 1$ and $ub_7 = 2$ ($t_6 = 1$, $t_7 = 2$), which are better upper bounds than the degrees.

Based on these propositions, we devise the UBP procedure (see Algorithm 2) to calculate an upper bound involving each vertex. Initially ub_i is set to $deg_G(i)$, then the upper bound of each vertex in U is improved according to Proposition 2 (lines 2-9). From line 10 to the end of Algorithm 2, the procedure aims at propagating the upper bound based on Proposition 3 until the upper bounds cannot be improved any more. The propagation procedure is guaranteed to converge as the upper bounds cannot be smaller than 0. Experiments in Section 6 show that, for both random and real-life large instances, UBP converges very fast, only in a limited number of iterations.

In both lines 7 and 14, we use binary search to find, for a given set I of integers, the maximum element $x \in I$ such that there are at least x integers in I that

are larger than or equal to x. The procedure works as follows: first, I is sorted by decreasing order, then, an iteration starts by comparing the middle element with its index in S (i.e., its position in the sorted list). If the middle element is greater (respectively lesser) than its index, the next iteration proceeds with the second half (respectively the first half) of I. This binary search procedure based on dichotomy performs at most $\log_2(|I|)$ operations.

Actually, we can also tighten the initial upper bound involving each vertex in V by repeating the process in lines 2-9 after replacing U with V before the propagating procedure (lines 10-17) starts. However, this procedure requires considerable memory especially for large graphs. The matrix representing $w_{ij}, (i,j) \in V \times V$ ($(i,j) \in U \times U$) requires a space of $O(|V|^2)$ ($O(|U|^2)$). Thus the overhead is not negligible. As a compromise, we set a threshold on the size of the vertex set. We apply the procedure of lines 2-9 to improve the upper bound involving each vertex only when the cardinality of the vertex set (U or V) is less than the threshold. In the following experiments, the threshold has empirically been set to 30000.

Algorithm 2: Upper bound propagation procedure

Input: Graph instance $G = (U, V, E)$
Output: An upper bound vector ub for each vertex in G.

```
1  ∀i ∈ U ∪ V, ub_i ← deg_G(i)
2  ∀(i,j) ∈ U × U, w_ij ← 0
3  for k ∈ V do
4      for (i,j) ∈ N(k) × N(k) do
5          w_ij ← w_ij + 1
6  for i ∈ U do
7      Binary search for the largest integer y_i such that |{j ∈ U : w_ij ≥ y_i}| ≥ y_i
8      if y_i < ub_i then
9          ub_i ← y_i
10 stable ← false
11 while stable ≠ true do
12     stable ← true
13     for i ∈ U ∪ V do
14         Binary search for the largest integer t_i such that
           |{j ∈ N(i) : ub_j ≥ t_i}| ≥ t_i
15         if t_i < ub_i then
16             ub_i ← t_i
17             stable ← false
18 return ub
```

To see how tight the upper bounds provided by UBP are, consider the example of Figure 1, the upper bounds achieved by UBP are $1, 2, 2, 1, 1$ for vertices of

U and $1, 2, 2, 2, 1$ for vertices of V. These upper bounds are actually all tight.

4.2 Combining UBP with BBClq: ExtBBClq

As UBP is independent of the search algorithm, we use it as a pre-processing procedure for BBClq to obtain an extended version named ExtBBClq. In ExtBBClq, we use the same branching heuristic as in the original BBClq algorithm: the vertex of the minimum degree in C_A is given the highest priority for branching. To efficiently implement ExtBBClq, we sort the arrays $N(i)$ ($\forall i \in U \cup V$) in ascending order of index number before the beginning of BBClq, so that the intersection operation in line 11 (Algorithm 1) can be accomplished in $O(|C_B| * log(|N(i)|))$ asymptotic time by binary search. More importantly, to make use of the upper bound information calculated by UBP, in ExtBBClq, instead of calculating the upper bound by calling the upper bound estimation method (i.e., $upper\_bound(A \cup \{i\})$ in line 9, we use the pre-computed ub_i returned by UBP as the upper bound in the current branch. Given that ExtBBClq is an enumeration algorithm, it has an exponential complexity.

5 Improving ILP formulation with UBP

In this section, we first recall the mathematical formulation of MBBP introduced in [5]. Then we elaborate our proposed formulation that takes advantage of the UBP procedure presented in the last section and define new valid inequalities to further tighten these formulations.

5.1 Formulation of MBBP from [5]

The following integer linear programming formulation for MBBP was proposed in [5], which relies on the complement bipartite graph of the input graph.

$$\max \quad \omega(G) = \sum_{i \in U} x_i \tag{1}$$

subject to:

$$x_i + x_j \leq 1, \forall \{i, j\} \in \bar{E} \tag{2}$$

$$\sum_{i \in U} x_i - \sum_{j \in V} x_j = 0 \tag{3}$$

$$x_i \in \{0, 1\}, \forall i \in U \cup V \tag{4}$$

where each vertex of $U \cup V$ is associated to a binary variable x_i indicating whether the vertex is part of the biclique, $\bar{E}$ is the set of edges in the complement bipartite graph of G. Constraint (2) requires that each pair of non-adjacent vertices cannot be selected at the same time (i.e., the solution must form a biclique). Constraint (3) enforces that the biclique is balanced. This formulation contains $|\bar{E}| + 1$ inequalities.

5.2 A new ILP formulation with UBP for MBBP

The above formulation contains $|\bar{E}| + 1$ inequalities (constraints). As a result, it is not applicable to solve large sparse graphs with ILP solvers like CPLEX. For example, a random sparse graph with 1000 vertices in each partition and an edge density of 0.1 has an average of 900,000 constraints. In real-life applications like those used in Section 6, the number of resulting constraints could be even much higher. To cope with this difficulty, we below propose a new formulation that relies on the initial graph and involves significantly fewer constraints for sparse graphs.

$$\max\ \omega(G) = z \tag{5}$$

subject to:

$$z = \sum_{j \in U} x_j \tag{6}$$

$$z = \sum_{i \in V} x_i \tag{7}$$

$$x_i z \leq \sum_{j \in N(i)} x_j, \quad \forall i \in U \cup V \tag{8}$$

$$x_i \in \{0, 1\}, \quad \forall i \in U \cup V \tag{9}$$

$$z \in \mathbb{Z}_+ \tag{10}$$

In the formulation, z is the half-size of the balanced biclique in the bipartite graph $G = (U, V, E)$. x_i is a binary variable associated to vertex i, indicating whether the corresponding vertex is part of the balanced clique. Constraint (8) is a quadratic inequality that requires the solution to be a biclique. As we want a linear model, we replace the quadratic inequality with the linear inequality:

$$z - (1 - x_i)\ell \leq \sum_{j \in N_G(i)} x_j \tag{11}$$

where ℓ is a large value bounding z. (Since we have an UBP procedure, we use $\ell = \max_{i \in U} ub_i$ in our computational experiments in Section 6).

It is clear that the new ILP model involves only $2+|U|+|V|$ constraints, which are independent from the number of edges in G. For our previous example of a random sparse graph with 1000 vertices in each partition and an edge density of 0.1, the new model contains only 2002 constraints, which is drastically less than the 900,000 constraints required by the model of [5].

5.3 Tightening the ILP formulations

The two formulations can be further tightened by valid inequalities. Let ℓ be a positive integer, and let $S^\ell \subseteq U$ (or $S^\ell \subseteq V$) be the set of all the vertices in U such that $ub_i \leq \ell$ for all $i \in S^\ell$. The following inequality is valid for MBBP:

$$\sum_{i \in S^\ell} x_i \leq \ell$$

This inequality indicates that the vertices in S^ℓ can only be part of a balanced clique with half-size less than ℓ. Moreover, for each vertex $j \in S^\ell$, we can lift the term associated with x_j:

$$(\ell - ub_j + 1)x_j + \sum_{i \in S^\ell \setminus \{j\}} x_i \leq \ell \quad \forall j \in S^{\ell-1}$$

The previous inequality can be extended as follows. Let T^ℓ be any maximal neighbor-disjoint subset of S^ℓ such that for all i and $j \in T^\ell$, $N(i) \cap N(j)$ is empty. The term 'maximal subset' means that no vertex $i \in S^\ell$ can be added to T^ℓ. Then we can derive the following valid inequality, where Ω^ℓ is the the collection of all the maximal neighbor-disjoint subsets of S^ℓ

$$\sum_{j \in T^\ell} (\ell - ub_j + 1)x_j + \sum_{i \in S^\ell \setminus T^\ell} x_i \leq \ell \quad \forall T^\ell \subseteq \Omega^\ell$$

Or equivalently,

$$\sum_{j \in T^\ell} (\ell - ub_j)x_j + \sum_{i \in S^\ell} x_i \leq \ell \quad \forall T^\ell \subseteq \Omega^\ell \tag{12}$$

In the following, we tighten the aforementioned formulations of MBBP by adding these inequalities. However, given a valid upper bound ℓ, Ω can be made of an exponentially large number of maximal neighbor-disjoint subsets of S^ℓ, since finding a maximal neighbor-disjoint subset in S^ℓ is equivalent

to finding a maximal stable set in the auxiliary graph $G' = (S^\ell, E')$ ($E' = \{\{i,j\} : i,j \in S^\ell, N(i) \cap N(j) \neq \emptyset\}$). To balance the tightness of the extended formulation and computational overhead of solving it, we set $\ell = \max_{i \in U} ub_i$ (i.e., $S^\ell = U$) and only generate $|U|$ inequalities by Algorithm 3. Each new inequality is associated with a maximal neighbor-disjoint subset seeding with vertex $i \in S^\ell$.

Algorithm 3: Generating valid inequalities defined in (12).

Input: Graph instance $G = (U, V, E)$, a valid upper bound ℓ, upper bound ub_i associated to each vertex $i \in U$.

Output: $|U|$ valid inequalities

```
1 S^ℓ ← {i ∈ U : ub_i ≤ ℓ}
2 for i ∈ S^ℓ do
3     T^ℓ ← {i}
4     for j ∈ S^ℓ and j ≠ i do
          /* w_jk is the number of common adjacent vertices between j
             and k, see Sec.2                                          */
5         if ∀k ∈ T^ℓ, w_jk = 0 then
6             T^ℓ ← T^ℓ ∪ {j}
7     Generate an inequality of (12) with T^ℓ and S^ℓ
```

One may observe that identical valid inequalities seeding with two vertices i and j of S^ℓ may be generated, but this is not an issue since modern solvers remove duplicate constraints automatically during pre-solving. The time complexity of Algorithm 3 is bounded in $O(|U|^3)$ when $S^\ell = U$. It is not hard to see that Inequalities (12) also hold for $S^\ell \subseteq V$. Setting $\ell = \max_{i \in V} ub_i$, we also generate $|V|$ inequalities associated with maximal neighbor-disjoint subsets of V.

Naturally, the lower ub_i is, the tighter these inequalities are. Consider the example of Figure 1, since the upper bound involving each vertex is given by UBP, we can produce the following valid inequalities ($\ell = 2$):

- Vertex 1 (and also 4) leads to $2x_1 + x_2 + x_3 + 2x_4 + x_5 \leq 2$
- Vertex 5 leads to $2x_1 + x_2 + x_3 + x_4 + 2x_5 \leq 2$
- Vertex 6 leads to $2x_6 + x_7 + x_8 + x_9 + x_{10} \leq 2$
- Vertex 10 leads to $x_6 + x_7 + x_8 + x_9 + 2x_{10} \leq 2$

The LP relaxation of the original formulation (1)-(4) yields an objective of 2.5 and nearly all the variables are fractional. Adding these four inequalities yields an objective of 2 and an integer solution, which proves to be optimal.

6 Computational experiments

This section is dedicated to a computational evaluation of the proposed algorithms for MBBP. All the experiments were conducted on a computer with an Intel Xeon© E5-2670 processor (2.5GHz and 2GB RAM) running CentOS 6.5. The algorithms are implemented in C++ and compiled with g++ using optimization option `-O3` [1].

To make a comprehensive assessment, we used both random bipartite graphs and large bipartite graphs from real world complex networks.

6.1 Performance on random graphs

Random graphs are commonly served as benchmark instances in the literature [7,20]. In this type of graphs, every possible edge occurs independently with fixed probability. For each graph, there are n vertices in each vertex set (*i.e.*, $n = |U| = |V|$) and the probability that an edge exists between a pair of vertices $(u, v) \in U \times V$ is p ($p \in [0, 1]$). A theoretical analysis in [5] showed that the maximum half-size of the balanced bicliques in such graphs is in the range $[\frac{\ln n}{\ln(1/p)}, \frac{2 \ln n}{\ln(1/p)}]$ with high probability (when n is sufficiently large).

The evaluation on random graphs is composed of two parts. The first part concerns the B&B algorithms and ILP formulations with or without additional inequalities. The second part involves evaluations of a more sophisticated method, the Branch-and-Cut (B&C) algorithm, which integrates formulations and generates valid inequalities on demand.

6.1.1 Results of B&B algorithms and ILP formulations

We compare the performance of six algorithms including both the existing approaches and the new approaches proposed in this work. The first two algorithms are B&B algorithms while the last four are different mathematical formulations solving with CPLEX 12.6.1.

- **BBClq**: a trimmed version of the algorithms presented in [9] for the more general problem where G might not be bipartite, without the bit-parallel technique [12] and the operation of bit-vector intersection.
- **ExtBBClq**: the extended version of BBClq combining UBP as presented in Section 4.2.
- **IP0**: the mathematical formulation proposed in [5].

[1] The code is available at: `https://github.com/joey001/mbbp_exact`.

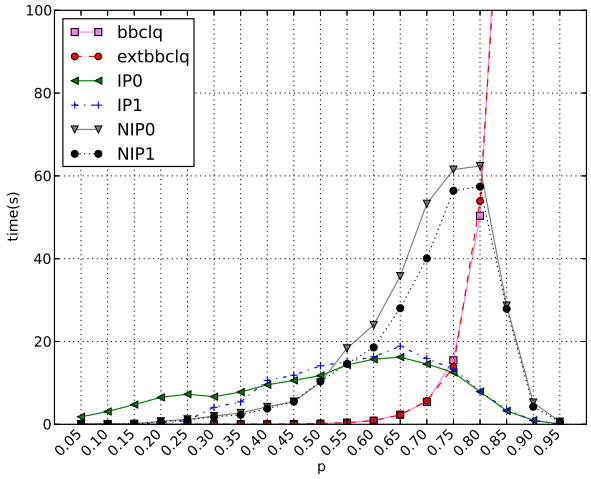

Fig. 2. Average computation times of the compared algorithms against different edge densities.

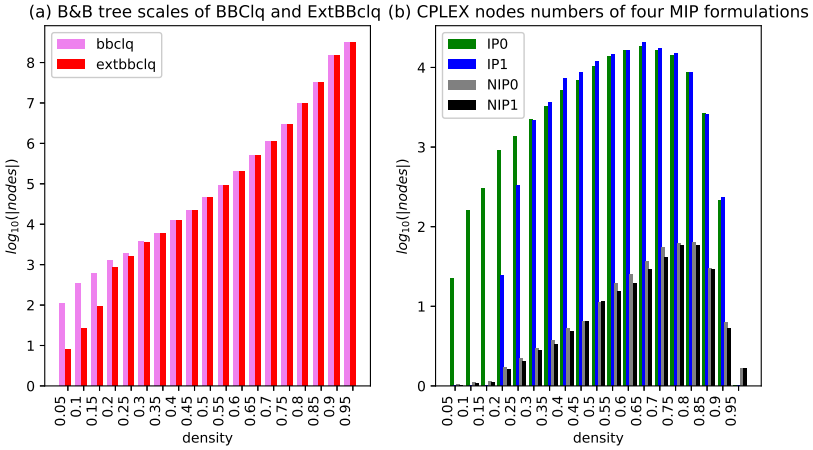

Fig. 3. Average B&B tree nodes of the compared algorithms on random graphs against different edge densities.

- **IP1**: The IP0 formulation enhanced with additional valid inequalities of Section 5.3.
- **NIP0**: The new mathematical formulation proposed in Section 5.
- **NIP1**: The NIP0 formulation enhanced with additional valid inequalities in Section 5.3.

We first generate 30 instances for each configuration of $n = 50$ and $p \in \{0.05, ..., 1\}$. Figures 2 and 3 summarize the performance of the aforementioned algorithms on these instances. One can observe that both BBClq and ExtBBClq are quite competitive when the graph density is less than 0.75. However, the time consumption increases exponentially with the increase of the graph density. Indeed, BBClq spends 165, 819 and 1806 seconds while ExtBBClq spends 154, 942 and 1790 seconds when the densities are 0.85, 0.90

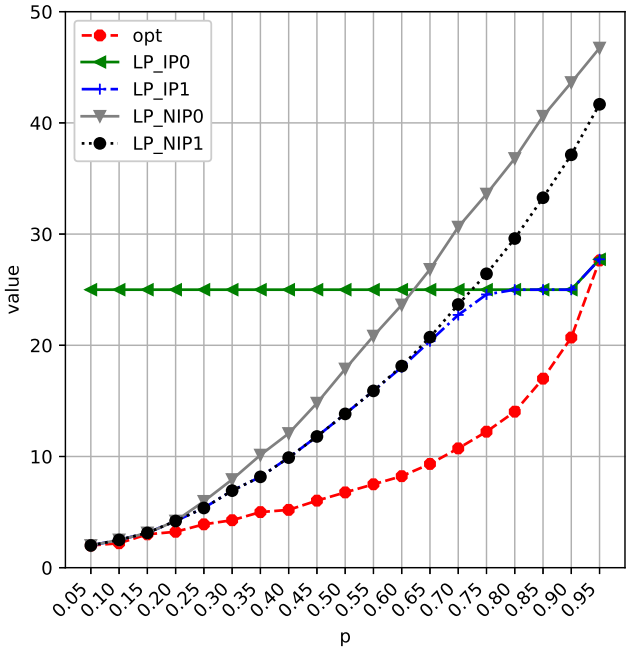

Fig. 4. Average upper bounds of linear relaxation against different edge densities.

and 0.95 respectively. On very dense graphs, ExtBBClq performs equally well as BBClq. These observations are also confirmed by the number of B&B tree nodes they enumerate in Figure 3. Indeed, we observe that ExtBBClq has a significantly smaller number of B&B tree nodes than BBClq only when the density is less than 0.35.

As for the four ILP formulations (IP0, IP1, NIP0, NIP1), when $p \leq 0.55$, NIP0 and NIP1 perform better than IP0 and IP1; otherwise, the reverse is true. In fact, when $p > 0.55$, the computation time of NIPs increases dramatically until p is around 0.8, and then drops beyond this density. However, from Figure 3, we observe that the new formulations lead to significant fewer B&B tree nodes with only one exception when $p = 0.95$. Concerning the valid inequalities, the performance is generally better with these valid inequalities than without them, especially when the graphs are not very dense.

Meanwhile, for all the tested random graphs, the time consumption of UBP is insignificant with respect to the whole search time (less than 0.01 seconds) for these instances. The number of iterations for propagating upper bounds is also trivial (closely around 3 for all the configurations).

Finally, Figure 4 shows a comparison of the linear relaxations of four ILP formulations (LP_IP0, LP_IP1, LP_NIP0 and LP_NIP1 correspondingly). In terms of the gap between LP_IP0 and LP_NIP0, when $p \leq 0.6$, the later is better than the first one; otherwise LP_IP0 is better. Indeed, the gap between

LP_IP0 (or LP_IP1) and optimal values is quite small for very dense graphs ($p \geq 0.95$). This observation generally matches the above conclusion that NIP performs better when $p \leq 0.55$ while IP0 and IP1 perform very well for dense graphs. Meanwhile, one notices that the formulations with the valid inequalities always lead to better or equal upper bounds. Actually, LP_IP1 with our valid inequalities can even achieve better bounds than LP_NIP0 and LP_NIP1.

6.1.2 Branch-and-cut

The valid inequalities introduced in this paper open the way to a Branch-and-Cut (B&C) algorithm implementation, where only necessary cuts would be generated. A key issue in designing such an algorithm is how to generate valid inequalities on demand, or find hyperplanes to separate the current fractional solution (this problem is called the separation problem [16]). In our case, by setting $\ell = \max_{i \in U} ub_i$ (*i.e.*, $S^\ell = U$), the problem of separating inequality (12) (Section 5.3) can be stated as follows.

The separation problem: Given a weighted bipartite graph $G = (U, V, E, f)$ with vertex weight function $f(i) = (\ell - ub_i)x_i^*$ for $i \in U \cup V$, x_i^* is the optimal solution of the linear relaxation. Does there exist a vertex-weight neighbor-disjoint subset $T^\ell \subseteq U$ such that $\sum_{j \in T^\ell} f(j) > \ell - \sum_{i \in U} x_i^*$?

One notices that the separation problem is equivalent to detecting the maximum weight stable set from a general graph $G' = (U, E', f)$ where an edge $\{i, j\}$ exists in E' if i and j have common neighbors in G (*i.e.*, $w_{ij} > 0$). It is well known that the maximum weight stable set problem in general graphs is NP-hard [21], and thus unlikely to be solved in polynomial time. However, there are many heuristic algorithms for the maximum stable set problem that can be used to separate the violated inequalities. In our case, we adapt a fast heuristic from [11] to find large stable sets. The algorithm works as follows. In each round, we initialize set $\mathcal{I}$ with a vertex from U and find a maximal vertex-weight neighbor-disjoint subset including $\mathcal{I}$. To achieve this, we consistently take a vertex of maximum weight from $\{i : \forall j \in \mathcal{I}, w_{ij} = 0\}$ and add it to $\mathcal{I}$ until no such vertex exists. Then, we test if the associated inequality is violated and proceed to the next round.

Moreover, in our B&C implementation, we select NIP0 as the base formulation since it incorporates less constrains. Noticing that the performances of NIP0 and IP0 complement each other on random graphs, we add the inequalities of IP0 for graphs of edge density larger than 0.55. We also limit the number of cuts to $\lceil 0.6|U \cup V| \rceil$ per search node (with reference to the successful B&C algorithm in [3]). The results of the B&C algorithm together with the

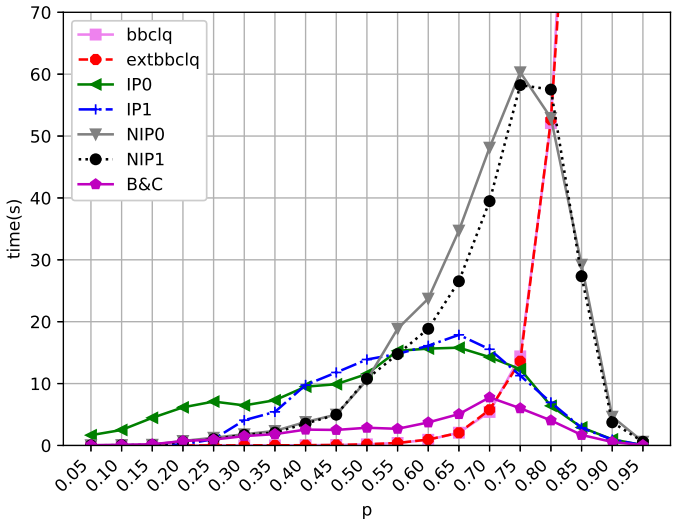

Fig. 5. Comparison of B&C with the other methods in terms of average computation times against different edge densities.

other methods in terms of average computational times for random graphs are presented in Figure 5. For graphs with $p \geq 0.45$, B&C dominates all above ILP formulations, including those with pre-added valid inequalities. For graph with densities smaller than 0.45, the computational times of B&C are comparable or more favorable than that of the underlying formulation NIP0. In a nutshell, this experiment shows that a further improvement can be achieved by combining the formulations and generating valid inequalities heuristically.

6.2 Performance on large real-life instances

In this section, we show experimental results to evaluate the performance of exact algorithms (formulations) on very large real-life networks. The data set includes 30 bipartite networks from the Koblenz Network Collection (KONECT) [8], which contains hundreds of networks derived from real-life applications, including social networks, hyperlink networks, authorship networks, physical networks, interaction networks and communication networks. The underlying applications do not necessarily require the determination of a maximum balanced biclique. However, these instances are interesting for our evaluation purpose since they are relevant representative examples of real-life bipartitite graphs, whose main features are summarized in Table 1, including the number of vertices and edge density. Irrelevant information for solving MBBP, such as multiple edges, vertex or edges weights have been ignored.

6.2.1 LNIP: adaption of existing formulations

We indicated that the new formulation of Section 5, which relies on the number of vertices of the graph, has much fewer constraints compared to the previous formulation [5]. On the other hand, our preliminary experiments disclosed that, since many KONECT instances are sparse and very large, most of these instances cannot be solved by CPLEX even with the new formulation with or without the additional valid inequalities (NIP0 and NIP1). Indeed, given a time limit of 3 hours, only 3 of the 30 selected instances (i.e., unicodelang, moreno_crime and jester1) can be solved on our computing platform.

To improve this situation, we devise a decomposition approach that divided the initial problem into several sub-problems and then solve each problem by NIP0. Suppose we have a vertex $k \in U$, and $V_k = N_G(k)$, $U_k = \cup_{i \in V_k} N_G(i)$, then the maximum balanced biclique including vertex k must be contained in $G_k[U_k, V_k]$. Thus, the following mathematical formulation can be introduced to solve MBBP.

$$\max_{k \in U} \max z_k \tag{13}$$

subject to:

$$x_k = 1 \tag{14}$$

$$z_k = \sum_{j \in U_k} x_j \tag{15}$$

$$z_k = \sum_{i \in V_k} x_i \tag{16}$$

$$z_k - (1 - x_i) ub_k \leq \sum_{j \in N_{G_k}(i)} x_j \quad \forall i \in U_k \cup V_k \tag{17}$$

$$x_i \in \{0, 1\}, \quad \forall i \in U_k \cup V_k \tag{18}$$

$$z_k \in \mathbb{Z}_+ \tag{19}$$

This formulation (called LNIP0) can be considered as a refinement of the aforementioned NIP0. In this formulation, z_k is the largest balanced biclique size including vertex k. LNIP0 allows us to decompose the input graph into several subgraph G_k ($k \in U$) and solve MBBP in each subgraph G_k independently, the overall best solution being the optimal solution in the initial graph. Indeed, for each vertex k, if its associated upper bound ub_k is smaller than or equal to the current best solution, we can save the effort of solving MBBP in subgraph G_k since no better solution can exist in G_k. Also, according to our observation, it is not worth using the valid inequalities of Section 5.3 for solving KONECT instances. This can be explained in a straight-forward way. After decomposition, each subgraph G_k becomes easy and can be solved effectively with NIP0, while calling UBP to generate additionnal valid inequalities for G_k (the time complexity is bounded by $O(|U|^4)$ in this case) will only yield an unnecessary overhead.

6.2.2 Experimental results of KONECT instances

We show the computational performances of BBClq, ExtBBClq and LNIP0 for KONECT instances in Table 1. A cut-off time limit of 3 hours (10800 seconds) is given to each algorithm. Column "instance", "$|U|$","$|V|$","density" indicates the general information of the graphs. Column "BEST" shows the best half-size found by all algorithms. An extra "*" indicates the optimality of this best value. Column "time" reports the pre-processing time (the time of running UBP) and column "iter" shows the average number of `while` loops (i.e., lines 11-17, Algorithm 2) needed to stabilize the upper bound of all the vertices. For each algorithm, we report the average time consumed to solve the corresponding instance (the time for pre-processing is also included). The shortest times among the three approaches are highlighted by bold font. If the instances cannot be solved within 3 hours, we show the best lower bound and upper bound in brackets. However, for instances actor-movie, dbpedia-team and gottron-trec, CPLEX still fails due to the memory limitation.

First of all, the time consumption of UBP is still insignificant with respect to the whole search time. The number of iterations for propagating upper bounds never exceed 100. For relatively small graphs like the first four cases in Table 1, the original BBClq algorithm performs equally well or even better than ExtBBClq and LNIP0 in terms of computation time. However, the extended algorithm ExtBBClq and new formulation LNIP0 outperform BBClq for the remaining very large instances. For example, for large instances like dbpedia-writer, dbpedia-starring, dbpedia-producer, ExtBBClq and LNIP0 even improve the computation time with four orders of magnitude. LNIP0 fails to achieve the optimal solution for some instances, like github, bookcrossing_full-rating and stackexchange-stackoverflow while performs well on most other instances. LNIP0 can be considered as a complementary algorithm to ExtBBClq given that LNIP0 performs significantly better than ExtBBClq for jester1, dblp-author.

In Figure 6, we compare the number of B&B tree nodes of BBClq and ExtBBClq for the 21 instances that can be solved by both algorithms in 3 hours. One can find that ExtBBClq enumerates fewer tree nodes for all the real-life instances, which confirms that the upper bounds actually help to reduce the search tree. The gain is quite significant for dbpedia-producer, dbpedia-writer and moreno_crime, where ExtBBClq prunes more than half of the nodes compared to BBClq.

Table 1
Computational results of the 3 approaches for KONECT instances. Instances are listed in increasing order of the number of vertices.

instance	$\|U\|$	$\|V\|$	density ($\times 10^{-4}$)	BEST	UBP		Computation time (s)		
					time (s)	iter	BBClq	ExtBBClq	LNIP0
unicodelang	254	614	8.047	4*	0.02	5	**0.01**	0.02	0.04
moreno_crime_crime	829	551	3.231	2*	0.05	3	**0.05**	0.06	0.06
opsahl-ucforum	899	522	71.855	5*	0.09	10	0.18	**0.13**	4.65
escorts	10106	6624	0.756	6*	5.69	6	**7.68**	10.05	17.94
jester1	73421	100	563.376	100*	1.86	1	1204.64	1123.66	**188.67**
pics_ut	17122	82035	1.637	27	21.84	7	[27-]	[23-]	[26-79]
youtube-groupmemberships	94238	30087	0.103	12*	0.96	21	222.88	**11.49**	407.10
dbpedia-writer	89356	46213	0.035	6*	0.19	12	283.16	0.35	**0.27**
dbpedia-starring	76099	81085	0.046	6*	1.07	31	530.61	4.67	**1.19**
github	56519	120867	0.064	12*	1.01	16	677.72	**150.66**	[10-45]
dbpedia-recordlabel	168337	18421	0.075	6*	24.33	7	214.45	24.67	**25.09**
dbpedia-producer	48833	138844	0.031	6*	0.27	11	535.44	0.62	**0.43**
dbpedia-location	172091	53407	0.032	5*	0.18	8	633.98	0.52	**0.26**
dbpedia-occupation	127577	101730	0.019	6*	0.27	8	909.03	**1.29**	7.35
dbpedia-genre	258934	7783	0.230	7*	3.94	9	171.86	**5.83**	11.13
discogs_lgenre	270771	15	1021.199	15*	0.06	1	37.08	**1.01**	9.84
bookcrossing_full-rating	105278	340523	0.032	13*	5.11	33	3102.66	**426.37**	[8-41]
flickr-groupmemberships	395979	103631	0.208	36	47.37	36	[34-]	[36-]	[22-147]
actor-movie	127823	383640	0.030	8*	5.54	27	6533.01	**1671.29**	-
stackexchange-stackoverflow	545196	96680	0.025	9*	4.62	29	4107.56	**265.8**	[6-25]
bibsonomy-2ui	5794	767447	0.575	8*	1.56	7	491.36	**13.84**	33.96
dbpedia-team	901166	34461	0.044	6*	3.08	15	2982.24	**241.06**	-
reuters	781265	283911	0.273	39	611.76	61	[35-]	[39-]	[19-192]
discogs_style	1617943	383	38.868	38	17.42	22	[23-]	[38-]	[37-93]
gottron-trec	556077	1173225	0.128	83	549.21	35	[33-]	[38-]	-
edit-frwiktionary	5017	1907247	0.773	19*	9.56	9	944.21	**152.5**	[16-36]
discogs_affiliation	1754823	270771	0.030	26*	12.01	17	[1-]	**1688.95**	[9-117]
wiki-en-cat	1853493	182947	0.011	14*	7.67	20	[1-]	28.72	**22.27**
edit-dewiki	425842	3195148	0.042	40	93.68	23	[1-]	[40-]	[1-184]
dblp-author	1425813	4000150	0.002	10*	19.86	21	[1-]	403.16	**21.30**

7 Conclusions and future work

In this paper, we proposed new ideas for designing exact algorithms for the NP-hard Maximum Balanced Biclique Problem in bipartite graphs. We introduced the Upper Bound Propagation (UBP) procedure for the sake of estimating tight upper bound involving each vertex. UBP starts from the initial bound

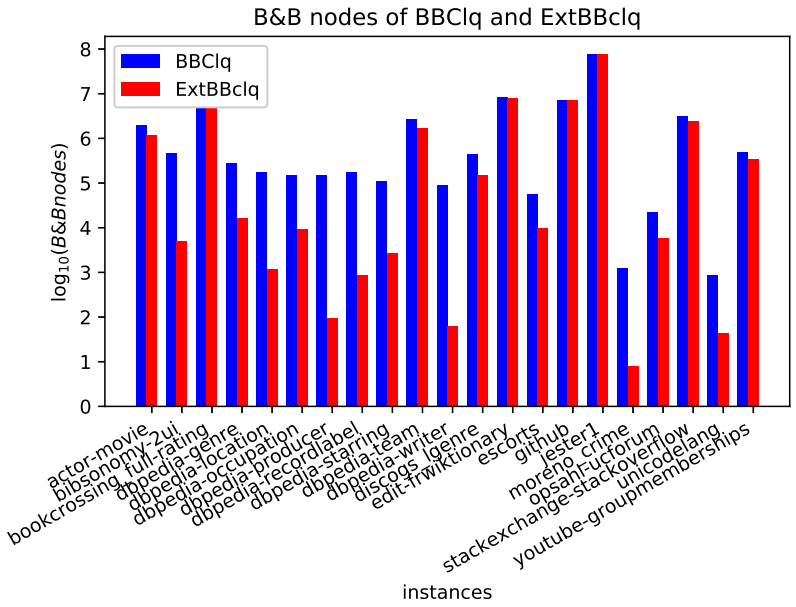

Fig. 6. The base-10 log scaled number of B&B tree nodes explored by the BBClq and ExtBBClq algorithms for the 21 solvable instances.

of each vertex and improves the upper bounds by a propagation procedure. We used UBP to enhance the existing B&B algorithm BBClq. Additionally, we proposed a new and more compact ILP formulation for MBBP that relies on the vertices of the input graph, contrary to the existing formulation that is based on the edges of the complement graph. We also devised a method of generating valid inequalities to tighten the MBBP formulations. Finally, we considered a Branch-and-Cut implementation that combines the new and existing formulations, as well as a fast clique-based heuristic to generate valid inequalities on demand. Our experiments on random bipartite graphs demonstrated the interest of the proposed ideas for creating effective B&B and B&C algorithms.

Based on these approaches, we then explored their applications on very large sparse networks from real-life applications (KONECT). Because of memory limitations, we proposed a decomposition scheme to exploit the new formulation to address the largest instances. Experiments also confirmed that new bounding techniques and new ILP formulations are two of the most competitive approaches for very large instances in this benchmark set.

For future work, it would be worthy of studying the bit-parallel technique [9,12] within our algorithms to further improve their performances. The idea of the upper bound propagation could also be adapted to other similar optimization problems.

Acknowledgment

We are grateful to the reviewers for their insightful comments which helped us to improve the paper. Support for the first author from the China Scholarship Council (2013-2017) is also acknowledged.

References

[1] A. A. Al-Yamani, S. Ramsundar, D. K. Pradhan, A defect tolerance scheme for nanotechnology circuits, IEEE Transactions on Circuits and Systems 54 (11) (2007) 2402–2409.

[2] N. Alon, R. A. Duke, H. Lefmann, V. Rodl, R. Yuster, The algorithmic aspects of the regularity lemma, Journal of Algorithms 16 (1) (1994) 80–109.

[3] B. Balasundaram, B. Sergiy, V.H. Illya, Clique relaxations in social network analysis: The maximum k-plex problem. Operations Research 59(1) (2011): 133-142.

[4] Y. Cheng, G. M. Church, Biclustering of expression data., in: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, vol. 8, 2000, pp. 93–103.

[5] M. Dawande, P. Keskinocak, J. M. Swaminathan, S. Tayur, On bipartite and multipartite clique problems, Journal of Algorithms 41 (2) (2001) 388–403.

[6] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.

[7] S. J. Hardiman, L. Katzir, Estimating clustering coefficients and size of social networks via random walk, in: Proceedings of the 22nd International Conference on World Wide Web, ACM, 2013, pp. 539–550.

[8] J. Kunegis, Konect: the koblenz network collection, in: Proceedings of the 22nd International Conference on World Wide Web, ACM, 2013, pp. 1343–1350.

[9] C. McCreesh, P. Prosser, An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem, in: Intl. Conference on CPAIOR, LNCS 8451, Springer, 2014, pp. 226–234.

[10] S. Ravi, E. L. Lloyd, The complexity of near-optimal programmable logic array folding, SIAM Journal on Computing 17 (4) (1988) 696–710.

[11] R.A. Rossi, D.F. Gleich, A.H. Gebremedhin, M.M.A. Patwary, Fast maximum clique algorithms for large graphs. in: Proceedings of the 23rd International Conference on World Wide Web. ACM, 2014, pp. 365–366.

[12] P. San Segundo, D. Rodríguez-Losada, A. Jiménez, An exact bit-parallel algorithm for the maximum clique problem, Computers & Operations Research 38 (2) (2011) 571–581.

[13] M. Soto, A. Rossi, M. Sevaux, Three new upper bounds on the chromatic number, Discrete Applied Mathematics 159 (18) (2011) 2281–2289.

[14] M. B. Tahoori, Application-independent defect tolerance of reconfigurable nanoarchitectures, ACM Journal on Emerging Technologies in Computing Systems 2 (3) (2006) 197–218.

[15] E. Tomita, T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, Journal of Global Optimization 37 (1) (2007) 95–111.

[16] L.A. Wolsey, Integer programming, Wiley (1998).

[17] Q. Wu, J.-K. Hao, A review on algorithms for maximum clique problems, European Journal of Operational Research 242 (3) (2015) 693–709.

[18] B. Yuan, B. Li, A low time complexity defect-tolerance algorithm for nanoelectronic crossbar, in: Proceedings of the International Conference on Information Science and Technology, IEEE, 2011, pp. 143–148.

[19] B. Yuan, B. Li, A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar, ACM Journal on Emerging Technologies in Computing Systems (JETC) 10 (3) (2014) 25.

[20] B. Yuan, B. Li, H. Chen, X. Yao, A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem, IEEE Transactions on Cybernetics 45 (5) (2015) 1040–1053.

[21] Y. Zhou, J.-K. Hao, A. Goëffon. PUSH: A generalized operator for the Maximum Vertex Weight Clique Problem. European Journal of Operational Research 257 (1) (2017) 41–54.