Tabu search with graph reduction for finding maximum balanced bicliques in bipartite graphs

Yi Zhou [a,b] and 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 Boulevard Lavoisier, 49045 Angers, France*

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

Eng Appl Artif Intell 77: 86–97, 2019
https://doi.org/10.1016/j.engappai.2018.09.017

Abstract

The Maximum Balanced Biclique Problem is a relevant graph model with a number of applications in diverse domains. However, the problem is NP-hard and thus computationally challenging. In this paper, we introduce a novel metaheuristic algorithm, which combines an effective constraint-based tabu search procedure and two dedicated graph reduction techniques. We verify the effectiveness of the algorithm on 30 classical random benchmark graphs and 25 very large real-life sparse graphs from the popular Koblenz Network Collection (KONECT). The results show that the algorithm improves the best-known results (new lower bounds) for 10 classical benchmarks and obtains the optimal solutions for 14 KONECT instances.

Keywords: Heuristics; clique problems; graph reduction; tabu search; complex networks.

1 Introduction

Let $G = (U, V, E)$ be a bipartite graph with disjoint vertex sets U, V and edge set $E \subseteq U \times V$. Informally, the *Maximum Balanced Biclique Problem* (MBBP) is to find in G the largest complete subgraph induced by two subsets of vertices (X, Y) $(X \subseteq U, Y \subseteq V)$ of equal size (see Section 2 for the formal

* Corresponding author.
Email addresses: `zhou.yi@uestc.edu.cn` (Yi Zhou),
`jin-kao.hao@univ-angers.fr` (Jin-Kao Hao).

definition). MBBP is known to be a NP-hard problem and thus computational challenging [10].

As a general graph model, MBBP has a number of relevant applications. For instance, in nanoelectronic system design, MBBP can conveniently formulate the problem of detecting $n \times n$ defect-free crossbars [2,22]. In this application, a crossbar is composed of two sets of orthogonal nanowires and a set of programmable switches. Each switch locates at an intersection of two nanowires. However, in the presence of defects, some nanowires or switches may be unusable. Thus, a key task in this application is to find the largest subset of $n \times n$ defect-free nanowires of the crossbar, such that the switches between any two orthogonal nanowires are functional. One observes that a crossbar can be represented by a bipartite graph $G = (U, V, E)$, where U and V correspond to two sets of orthogonal nanowires, an edge $\{i, j\} \in E$ corresponds to a functional switch between two orthogonal nanowires $i \in U$ and $j \in V$. As a result, the problem of detecting the largest $n \times n$ defect-free subset of the crossbar is equivalent to find the maximum balanced biclique from the bipartite graph representing the original crossbar. An example of this application is shown in Figure 1. Since defects usually appear rarely and randomly, bipartite graphs from such an application are typically dense random graphs. Other applications include computational biology where MBBP is used to simultaneously group genes and their expressions under different conditions (called biclustering) [6], and social network analysis where MBBP is a subproblem of co-clustering [15]. Unlike nanoelectronic system design, biology and social networks usually correspond to sparse graphs. Generally, the clique and biclique models are known to be useful tools for winner determination in combinatorial auctions [11,24], building efficient content pipelines for consumer Internet portal [9] and so on.

1.1 Literature review

Given the significance of MBBP as a NP-hard problem and its relevance in practice, a number of methods, including approximate, exact and heuristic algorithms have been proposed in the literature. For example, in [8], the relations between the approximate hardness of MBBP and 3-SAT as well as the maximum clique problem were established. In [19], a polynomial algorithm was given to find a balanced biclique with size $\lfloor \frac{\ln n}{\ln (2en^2/m)} \rfloor$ (the cardinality of $|X|$ or $|Y|$) for a graph with n vertices and m edges. In [22], a recursive exact algorithm for searching a maximum balanced biclique with a given size was proposed. However, the computational time of this algorithm becomes prohibitive when the number of vertices of the graph exceeds $(32, 32)$. Very recently, an upper bound propagation procedure was introduced and used to enhance a simple branch and bound (B&B) algorithm [32]. In [17], another ex-

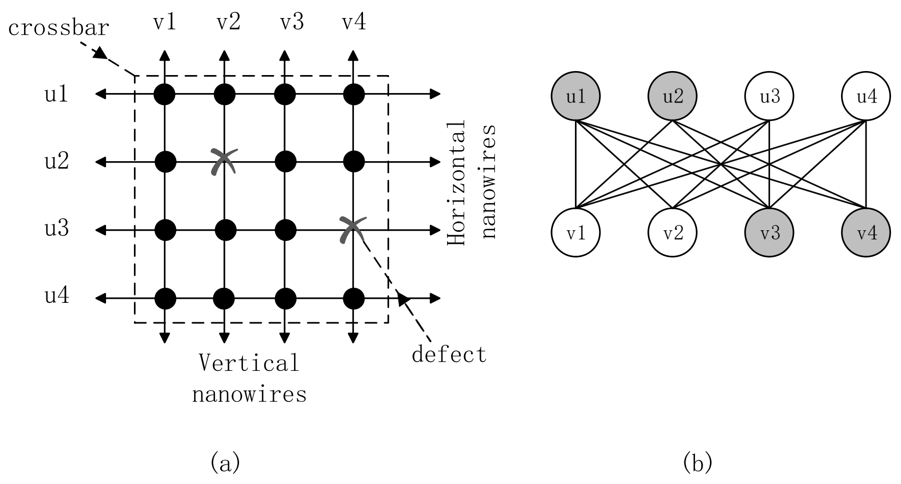

Fig. 1. (a) A 4×4 nanoelectronic crossbar with two defects. (b) A bipartite graph representing this crossbar.

act approach for MBBP for general (non-bipartite) graphs was studied, which follows a classical B&B algorithm for the popular maximum clique problem [25] with additional symmetry breaking techniques.

To cope with the computational challenge of MBBP, heuristic methods constitute an interesting approach. These methods aim to obtain satisfactory solutions in an acceptable time frame without guaranteeing the optimality of the attained solutions. The majority of existing heuristic algorithms solved the equivalent maximum balanced independent set problem for the bipartite complement, instead of directly seeking the maximum balanced biclique in the given graph. For example, [22] proposed a greedy algorithm based on vertex-deletion, which iteratively removes vertices with maximum degree from the bipartite complement until the set of remaining vertices forms an independent set. [2] presented an improved greedy heuristic, in which the vertex connecting the maximum number of vertices of minimum degree is removed. [27] introduced another greedy algorithm, which iteratively deletes vertices adjacent to the maximum number of vertices in a restricted set. [28] accelerated this algorithm by removing multiple vertices at each iteration. Recently, [29] proposed a powerful (and rather complex) evolutionary algorithm combining structure mutation and repair-assisted restart. The computational results showed that this algorithm performed very well on random dense graphs, which represent one type of the most challenging instances for MBBP.

1.2 Motivations and contributions

This work is motivated by the following observations. Our literature review shows that unlike the classical maximum clique problem and some of its variants for which numerous algorithms have been proposed [25], there are only few solution methods available for MBBP in bipartite graphs. Moreover, graphs from real-life applications like social networks are usually very large with millions, or even billions of vertices, rendering most existing approaches unpractical. This work aims to fill the gap by developing improved methods for MBBP in bipartite graphs, which should be able to handle both random dense graphs and large real-life networks. The contributions of this work are twofold.

First, we introduce an effective algorithm called “tabu search with graph reduction for MBBP (TSGR-MBBP)", which combines an original tabu search procedure and two dedicated graph reduction techniques. This is the first study mixing local optimization and graph reduction within the iterated search framework for MBBP. Generally, the proposed algorithm seeks maximum balanced bicliques directly on the given graph. Compared to the existing approaches which search for balanced independent sets on the complement graph, operating on the given graph requires much less memory for large sparse graphs. Specifically, TSGR-MBBP employs two bound-based reduction techniques to shrink progressively the given graph (Sections 3.5 and 3.6) and uses the dedicated tabu search procedure to effectively explore the search space of the reduced graph (Section 3.4).

Second, we carry out experiments on both random and real-life instances to demonstrate the effectiveness of the proposed algorithm. For the set of 30 random instances used in the literature, the algorithm dominates state-of-the-art algorithms including the best-performing heuristic of [29] and the powerful integer programming solver CPLEX. The algorithm attains 10 improved best solutions (i.e., new lower bounds) and matches the best-known results for the remaining 20 cases. For the 25 very large real-life instances from the well-known Koblenz Network Collection, the algorithm proves the optimal solutions for 14 instances (by obtaining equal upper and lower bounds) and reports tight lower bounds (better than those of CPLEX) for the remaining instances. We also analyze the impact of key algorithmic components to better understand the proposed algorithm (Section 5).

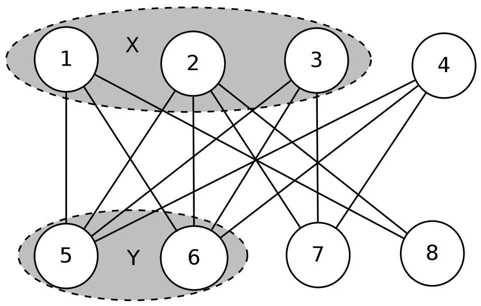

Fig. 2. $X = \{1, 2, 3\}$, $Y = \{5, 6\}$, $\mathcal{N}(X \cup Y) = \{4, 7, 8\}$, (X, Y) is a biclique of balanced size of 2 with a balance deviation of 1, while $(\{1, 2\}, \{5, 6\})$ is the maximum balanced biclique.

2 Preliminary

2.1 Definitions and notations

We first introduce the following definitions, supposing that $G = (U, V, E)$ is a bipartite graph, X is a subset of U ($X \subseteq U$), Y is a subset of V ($Y \subseteq V$) and $G[X \cup Y] = (X, Y, E(X \cup Y))$ is the *subgraph* induced by $X \cup Y$.

Definition 1: If $G[X \cup Y]$ is a complete bipartite graph, i.e., $E(X \cup Y) = X \times Y$, then $G[X \cup Y]$ is a *biclique* of G, which is also denoted by (X, Y). The biclique is balanced if $|X| = |Y|$.

Definition 2: Given a biclique (X, Y), its *balanced size* is $min(|X|, |Y|)$, and its *balance deviation* is $||X| - |Y||$. If the balance deviation is 0, (X, Y) is a balanced biclique of size $|X|$ $(= |Y|)$.

Definition 3 [10]: The Maximum Balanced Biclique Problem (MBBP) is to find a balanced biclique (X^*, Y^*) of maximum cardinality in G.

Let $\Omega(G)$ denote the search space composed of all balanced bicliques in G, we use Ω^k to represent the relaxed search space including all bicliques with a balance deviation no more than k ($k \geq 0$), i.e., $\Omega^k = \{(X, Y) : X \subseteq U, Y \subseteq V, E(X, Y) = X \times Y, ||X| - |Y|| \leq k\}$.

Given a biclique (X, Y) in Ω^k, its *quality* is measured by its balanced size $min(|X|, |Y|)$. For two bicliques (X_1, Y_1) and (X_2, Y_2), (X_1, Y_1) is better than (X_2, Y_2) if $min(|X_1|, |Y_1|) > min(|X_2|, |Y_2|)$.

Finally, we introduce the following notations, which are needed for the description of the proposed approach of Section 3.

- Given a vertex $i \in U \cup V$, $N(i)$ denotes the set of vertices adjacent to i,

i.e., $N(i) = \{j : \{i,j\} \in E\}$. Clearly, if $i \in U$, then $N(i) \subseteq V$, otherwise, $N(i) \subseteq U$.
- Given $S \subseteq U \cup V$, $\mathcal{N}(S)$ denotes the subset of vertices from $(U \cup V) \setminus S$ that are adjacent to at least one vertex in S, i.e., $\mathcal{N}(S) = (\bigcup_{i \in S} N(i)) \setminus S$.

Figure 2 illustrates the above notations and definitions with a bipartite graph composed of 8 vertices and 13 edges.

2.2 Linear programming formulations

In terms of computational complexity, the decision version of MBBP is NP-complete in the general case [3,10], even though the maximum biclique problem without the balance constraint is polynomially solvable by the maximum matching algorithm [6].

A simple Integer Linear Programming (ILP) model for MBBP can be found in [7]. Given $G = (U, V, E)$, the ILP model is presented as follows.

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

subject to:

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

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

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

where each vertex of U (V) is associated to a binary variable x_i (x_j) indicating whether the corresponding vertex is part of the biclique, $\bar{E}$ is the set of edges in the bipartite complement of G. Objective (1) maximizes the size of the biclique. Constraint (2) ensures that each pair of non-adjacent vertices cannot be selected at the same time (i.e., the solution must be a clique). Equation (3) enforces that the returned biclique is balanced.

3 Tabu search with graph reduction

The proposed TSGR-MBBP algorithm is based on the popular and general tabu search method [13], which has been used to solve other clique problems [24,25,31]. Like any general method, it is critical to suitably adapt the method to the considered problem. For our MBBP problem, we propose the constraint-based tabu search (Section 3.4), which explores the search space of slightly unbalanced bicliques. In order to help tabu search to avoid non-promising search regions, we introduce two bound-based graph reduction techniques to shrink progressively the given graph (Sections 3.5 and 3.6).

3.1 Rationale of the proposed approach

Many real-life networks have a large number of vertices but with very low edge density. Existing approaches for solving MBBP rely heavily on the complement graph and the adjacent matrix representation. Unfortunately, the complement of a massive graph usually results in very high memory consumption, making most of existing MBBP approaches unpractical. To avoid this difficulty, the proposed algorithm operates directly on the input graph, which is a memory-efficient way for processing very large sparse networks. From an algorithmic perspective, our algorithm iteratively seeks improved solutions by local search combined with graph reduction strategies. Specifically, the algorithm starts from an initial solution (a slightly relaxed balanced biclique) and uses move operators to improve the solution iteratively. However, we still need to answer a crucial question: how to improve the solution effectively while maintaining the two main constraints of a solution (balanced and biclique)?

Intuitively, local search operators that are successful for the maximum clique problem [25] can be applied to MBBP, such as “add" (adding a vertex to the solution), “swap" (exchanging a vertex in the solution with another vertex out of the solution), or even “drop" (dropping a vertex from the solution). However, given a balanced biclique, an application of any of these operators results in an unbalanced biclique. To cope with this difficulty, we propose to (slightly) relax the balance requirement and allow the algorithm to explore both balanced and slightly unbalanced bicliques. For this purpose, we adopt the general “push" operator initially designed for the maximum vertex weight clique problem [31] to explore solutions within the relaxed search space Ω^2 rather than Ω^0 (see Section 2.1).

Another key idea we used is graph reduction. Given a bipartite graph G and a known best balanced size ω (a lower bound), it is clear that to further improve ω, it is useless to consider any vertex of the graph whose degree is smaller than or equal to ω since such a vertex can in no way extend the best solution found so far. Consequently, the vertices with a degree smaller than or equal to ω along with the incident edges can be safely removed from the graph. Our algorithm integrates this idea to dynamically prune the graph under consideration, which proves to be highly effective on massive sparse graphs.

Finally, applying the graph pruning techniques can disconnect the original graph into several connected subgraphs. This observation can be explored advantageously to further reduce the graph in combination with an exact algorithm. Indeed, if a subgraph is small enough such that an exact algorithm can identify the *maximum* balanced biclique quickly, then the subgraph can be definitively removed since the subproblem (associated to the subgraph) is optimally solved. Moreover, the optimal solution of this subgraph can be used to update the current best balanced biclique (and the lower bound bound), which can lead to additional reduction of the graph.

3.2 General procedure of TSGR-MBBP

Algorithm 1: Main framework of TSGR-MBBP

Input: Graph instance $G = (U, V, E)$, tabu search depth L, cardinality threshold K for graph reduction with exact algorithm, tabu tenure parameter α

Output: The maximum balanced biclique.

```
1  begin
2      $(X^b, Y^b) \leftarrow (\emptyset, \emptyset)$;   /* The largest balanced biclique found so far */
3      $\omega = 0$ ;   /* The largest balanced size found so far */
4      while stopping condition is not met do
           // Find an improved biclique from a new initial biclique
5          $(X, Y) \leftarrow Init\_Solution(G)$ ;   /* Section 3.3 */
6          $(X, Y) \leftarrow Constraint\_Tabu\_Search(G, (X, Y), L, \alpha)$ ;   /* Section 3.4 */
7          if $min(|X|, |Y|) > \omega$ then
8              $(X^b, Y^b) \leftarrow (X, Y)$;
9              $\omega \leftarrow min(|X|, |Y|)$
           // Graph reduction procedure using improved balanced size $\omega$
10         while $\omega \geq min_{v \in U \cup V}\{|N(v)|\}$ do
               // First graph reduction
11             $G \leftarrow Peel(G, \omega)$ ;   /* Section 3.5 */
               // Second graph reduction
12             for each connected subgraph $G_i[U_i \cup V_i]$ in $G$ do
13                 if $|U_i| + |V_i| \leq K$ then
14                     $(X, Y) \leftarrow Exact\_Search(G_i, \omega)$ ;   /* Section 3.6 */
15                     if $min(|X|, |Y|) > \omega$ then
16                         $(X^b, Y^b) \leftarrow (X, Y)$;
17                         $\omega \leftarrow min(|X|, |Y|)$
18                     $G \leftarrow G[(U \setminus U_i) \cup (V \setminus V_i)]$
19         if $|U| \leq \omega \vee |V| \leq \omega$ then
20             return $Make\_Balance(X^b, Y^b)$ ;   /* $(X^b, Y^b)$ is an optimum solution */
21 return $Make\_Balance(X^b, Y^b)$
```

Our tabu search with graph reduction for MBBP (TSGR-MBBP) (Algorithm 1) explores the above ideas. TSGR-MBBP includes two main components: the Constraint-Based Tabu Search (CBTS) procedure and the graph reducing procedure. The CBTS procedure is used to find high quality bicliques in the relaxed search space Ω^2, while the graph reducing procedure aims to shrink progressively the current graph without losing optimal solutions.

After setting the best biclique (X^b, Y^b) to $(\emptyset, \emptyset)$ and the best balanced size ω to 0 (lines 2 and 3), the algorithm repeats the main 'while' loop (lines 4-20) until a stopping condition is met. For each 'while' loop, an initial biclique, which is not necessarily balanced, is first generated by Init_Solution() (line 5, see Section 3.3), and then further improved by the CBTS procedure (Constraint_Tabu_Search(), line 6, see Section 3.4). If the resulting biclique has a balanced size larger than the current best balanced size ω, the best biclique (X^b, Y^b) and the best balanced size are updated (lines 7-9).

Now, if the current best balanced size is greater than or equal to the degree of any vertex in the current graph, the graph reduction procedure is activated (lines 10-18). This procedure includes two phases: first, reducing the current graph by the *Peel* procedure (line 11, see Section 3.5); second, determining the optimal balanced size of each connected subgraph with up to K (a predefined parameter) vertices by a branch-and-bound (B&B) exact algorithm (Exact_Search(), line 14, see Section 3.6) and then deleting these subgraphs from the current graph (line 18). The optimal biclique of each subgraph is possibly used to update the recorded best biclique (lines 15-17). Finally, thanks to graph reduction, ω is proven to be the optimal balanced size if ω (which is a lower bound of the maximum biclique) is no less than the cardinality of any partition ($|U|$ or $|V|$, which is an upper bound) in the current graph (lines 19-20). In this case, the algorithm terminates immediately.

As explained in Section 2, the proposed algorithm operates on the relaxed biclique space Ω^2. As a result, the current solution (X, Y) and the best biclique found so far (X^b, Y^b) are not necessarily balanced with nevertheless a balance deviation of at most 2. Actually, the three procedures: Init_Solution(), Constraint_Tabu_Search() and Exact_Search() generate or return a biclique with a balance deviation that does not exceed 2. The procedure of retrieving a strict balanced biclique of size ω from an unbalanced biclique is accomplished by Make_Balance(). This procedure simply removes vertices from the larger set X^b or Y^b until a balanced biclique of size ω is obtained. Obviously, no more than 2 vertices will be removed by Make_Balance().

Finally, the whole algorithm terminates when a stopping condition is met. For the experiments reported in this paper (Sections 4 and 5), we use a cutoff time limit as our stopping condition.

3.3 Initial solutions

The Init_Solution() procedure is invoked to initialize each restart of TSGR-MBBP with a new biclique. This procedure starts from a trivial solution formed by a random vertex from $U \cup V$, say $(X, Y) = (\{1\}, \emptyset)$ (without loss of generality). Then, it iteratively expands the solution by alternatively adding one vertex v to the set X or Y, v being necessarily connected to all vertices of the other set. In the first iteration, a vertex is selected randomly from the set $\cap_{i \in X} N(i) \setminus Y$. In the next iteration, we switch to the set $\cap_{i \in Y} N(i) \setminus X$. The procedure continues until the current considered set becomes empty. The time complexity of this procedure is bounded by $O(|U \cup V| \times |E|)$.

Consider Figure 2 as an example and suppose that we start from solution $(X, Y) = (\{1\}, \emptyset)$, the procedure expands (X, Y) by adding to Y an arbitrary vertex from $N(1) \setminus \emptyset = \{5, 6, 8\}$ (say 5). Next iteration adds to X a random vertex from $N(5) \setminus \{1\} = \{2, 3, 4\}$. Suppose that the solution $(X, Y) = (\{1, 2, 3\}, \{5, 6\})$ is achieved after four iterations. Then the fifth iteration tries to expand Y with a vertex from $\mathcal{N}(X) \setminus Y$. This set being empty, the procedure stops and returns $(X, Y) = (\{1, 2, 3\}, \{5, 6\})$ as its output.

The biclique (X, Y) returned by this procedure may not be strictly balanced, but the balance deviation can never exceed 1. This biclique is served as the starting solution for the tabu search procedure which is explained below.

3.4 Constraint-Based Tabu Search

The Constraint-Based Tabu Search (CBTS) is the local optimization component of TSGR-MBBP, which explores bicliques in the relaxed space Ω^2.

3.4.1 General procedure of CBTS

Following the general principles of tabu search [13], CBTS (Algorithm 2) starts with an initial solution (X, Y) (provided by the procedure of Section 3.3) and then iteratively improves the incumbent solution. After initiating the iteration counter I, the best biclique found so far (X^*, Y^*) and the tabu list T (lines 2-3), CBTS enters the main 'while' loop (lines 4-26) to perform a number of iterations to improve the solution. At each iteration, the algorithm first builds a vertex set $C \subseteq \mathcal{N}(X \cup Y)$ that contains vertices of interest for solution transformation (line 5). Then it considers to expand the solution with a vertex from a *candidate set* $C_{expand} \subseteq C$ (lines 6-8). If no vertex can be used to expand the solution (i.e., $C_{expand} = \emptyset$), CBTS considers to swap a vertex of the solution with a vertex taken from another *candidate set* $C_{plateau} \subseteq C$ (lines 10-14).

Both the expanding and swapping operations are performed with the help of the "push" operator introduced in [31] and followed by a possible repairing operation if the allowable balance deviation is violated (lines 17-23). The best solution is updated at the end of the iteration if an improved solution is found (lines 24-25). During the search, a tabu list is used to record the vertices that leave the solution to prevent the search process from revisiting previously examined solutions (lines 3,14,20,23).

Algorithm 2: Constraint-Based Tabu Search

```
Input: Graph instance $G = (U, V, E)$ with $n = |U| + |V|$, starting solution $(X, Y)$,
       tabu search depth $L$, tabu tenure parameter $\alpha$.
Output: The best biclique $(X^*, Y^*)$ found.
1  begin
2      $I \leftarrow 0$, $(X^*, Y^*) \leftarrow (X, Y)$; /* $I$ is the iteration counter, $(X, Y)$ is the
       current solution, $(X^*, Y^*)$ keeps the best biclique found so far */
3      $T[1...n] \leftarrow [0...0]_n$ ;                        /* Initiate tabu list $T$ */
4      while $I \leq L$ do
           // Improve the current solution
5          Set $C = \{i \in N(X), Y \setminus N(i) \leq 1 \vee i \in N(Y), X \setminus N(i) \leq 1\}$ ;    /* When
           introducing any vertex of $C$ into $(X, Y)$, at most one vertex
           from the solution needs to be expelled */
6          Set $C_{expand} = \{i \in C : \delta_i > 1, min(|X|, |Y|) + 1 > min(|X^*|, |Y^*|)\}$ ;
           /* Any vertex of $C_{expand}$ can be added to expand the solution
           without needing to expel any vertex from the solution */
7          if $C_{expand} \neq \emptyset$ then
8              Pick randomly an eligible vertex $i \in C_{expand}$; $(X, Y) \leftarrow (X, Y) \oplus push(i)$
9          else
10             Set $C_{plateau} = \{i \in C : \delta_i = 0\}$ ;    /* Any vertex of $C_{plateau}$ can be
               swapped into $(X, Y)$ against a vertex of $X$ or $Y$ */
11             if $C_{plateau} \neq \emptyset$ then
12                 Pick randomly an eligible $i \in C_{plateau}$ ; $(X, Y) \leftarrow (X, Y) \oplus push(i)$ ;
13                 Let $j$ be the swapped out vertex;
14                 $T[j] \leftarrow I + tt(\alpha, |A|)$ ; /* $A = X$ if $j \in X$, $A = Y$ if $j \in Y$ */
15             else
16                 return $(X^*, Y^*)$ ;    /* CBTS terminates if both $C_{plateau}$ and
                   $C_{plateau}$ become empty */
           // Recover balance when the balance deviation exceeds 2
17         if $||X| - |Y|| > 2$ then
18             while $|X| > |Y|$ do
19                 Pick a random vertex $j \in X$ ; $X \leftarrow X \setminus \{j\}$ ;
20                 $T[j] \leftarrow I + tt(\alpha, |X|)$ ;  /* Update tabu tenure of the dropped
                   vertex */
21             while $|X| < |Y|$ do
22                 Pick a random vertex $j \in Y$ ; $Y \leftarrow Y \setminus \{j\}$ ;
23                 $T[j] \leftarrow I + tt(\alpha, |Y|)$ ;
           // Update the best solution
24         if $min(|X|, |Y|) > min(|X^*|, |Y^*|)$ then
25             $(X^*, Y^*) \leftarrow (X, Y)$
26         $I \leftarrow I + 1$
27 return $(X^*, Y^*)$
```

During the iteration, if both C_{expand} and $C_{plateau}$ are found to be empty, CBTS terminates and returns the best solution recorded in (X^*, Y^*) (line 16). Another (normal) stopping condition for CBTS is when the number of iterations I reaches the prefixed value L (called tabu search depth) (line 4).

Finally, note that during each iteration, the vertex selected from C_{expand} or $C_{plateau}$ must be an *eligible* vertex. A vertex is *eligible* if 1) it is not forbidden by the tabu list, or 2) pushing this vertex into the solution leads to a new biclique whose size is larger than the recorded best biclique found so far (X^*, Y^*). Condition 2 is the so-called *aspiration criterion* that revokes the forbidden status of a vertex.

3.4.2 Exploration of search space Ω^2 by solution transformation

To explore the search space Ω^2 and visit different bicliques, CBTS relies on the "push" operator that was initially proposed for the maximum weight clique problem in [31]. Generally, "push" operates with a *candidate set* composed of specific vertices that do not belong to the incumbent solution. To generate a new clique from the incumbent solution, the operator picks a vertex from the candidate set, pushes the vertex to the solution and then expels $p \geq 0$ vertices from the solution to maintain the feasibility of the new solution.

In the context of MBBP, given a biclique (X, Y) with $X \subseteq U$ and $Y \subseteq V$, let (X', Y') denote the new biclique obtained by pushing a selected vertex $i \in \mathcal{N}(X \cup Y)$ into the solution. We represent this transformation by $(X', Y') \leftarrow (X, Y) \oplus push(i)$ and use $\delta_i = min(|X'|, |Y'|) - min(|X|, |Y|)$ to denote the change of the balanced sizes between (X', Y') and (X, Y). Note that δ_i can be zero, negative and positive, respectively indicating that (X', Y') is of equal, worse, and better quality compared to (X, Y).

To identify an appropriate candidate set used by "push" for solution transformations, we build, with reference to the current solution (X, Y), a constrained set C from the vertices of $\mathcal{N}(X \cup Y)$ satisfying the following property: each vertex in C is adjacent to all the vertices of X (or Y), or all but one vertex of X (or Y). Formally, we specify the set C as follows.

$$C = \{i \in \mathcal{N}(X \cup Y) : i \in U \wedge |N(i) \cap Y| \geq |Y| - 1, i \in V \wedge |N(i) \cap X| \geq |X| - 1\} \tag{5}$$

To make the search more focused, we further identify, among the vertices of C, specific vertices satisfying additional properties to form two *restricted candidate sets* C_{expand} and $C_{plateau}$ that are used by the "push" operator. Specifically, C_{expand} is a subset of C such that pushing any vertex of this subset into the current solution leads always to a larger (better) biclique, while any vertex of $C_{plateau} \subseteq C$ can be exchanged with a vertex of the current solution without changing the biclique size. Formally, we specify the two restricted candidate sets C_{expand} and $C_{plateau}$ as follows.

$$\begin{aligned} C_{expand} &= \{i \in C : \delta_i > 0, min(|X|, |Y|) + 1 > min(|X^*|, |Y^*|)\} \\ C_{plateau} &= \{i \in C : \delta_i = 0\} \end{aligned} \tag{6}$$

where δ_i is calculated by the following rule (supposing $i \in \mathcal{N}(X) \setminus Y$ without loss of generality).

$$\delta_i \leftarrow \begin{cases} |Y \cap N(i)| - |Y| & \text{, if } |X| \geq |Y| \\ min(|X| + 1, |Y \cap N(i)|) - |X| & \text{, otherwise} \end{cases} \tag{7}$$

The correctness of Equation (7) can be checked simply. If $|X| \geq |Y|$, the balanced size of (X, Y) decrases from $|Y|$ to $|Y \cap N(i)|$; otherwise, the balanced size changes from $|X|$ to $min(|X| + 1, |Y \cap N(v)|)$. For vertex $i \in \mathcal{N}(Y) \setminus X$, the above equation holds when the roles of X and Y are exchanged. In our implementation, we maintain the values $|N(i) \cap Y|$ and $|N(i) \cap X|$ for all vertices, thus δ_i can be computed in constant time.

Given these restricted candidate sets C_{expand} and $C_{plateau}$, CBTS naturally explores with priority C_{expand} since pushing vertices of this set always improves the solution. Only when C_{expand} is empty, $C_{plateau}$ is used to explore bicliques of equal size. After pushing a vertex of $C_{plateau}$ into the solution, the expelled vertex j is marked tabu for the following $tt(\alpha, |A|)$ ($A = X$ if $u \in X$, otherwise $A = Y$) iterations. According to [31], the tabu tenure $tt(\alpha, l)$ is defined by the function: $tt(\alpha, l) = max(7, \alpha * random(l))$ where α (a non-negative real number) is a predefined parameter and $random(l)$ returns a random integer in $[0, l]$. Note that vertices dropped from the solution during balance repairing (lines 17-23) are also classified tabu, thus preventing these vertices from joining again the solution during the period fixed by the tabu tenure.

Finally, to implement the tabu list effectively, we employ an integer vector T of length n ($n = |U \cup V|$), whose elements are initially set to zero. During the search, each time a vertex j is expelled from the solution by “push", $T[j]$ is set to the current iteration number I plus the tabu tenure tt (lines 14,20, 23 Algorithm 2). As such, the tabu status of a vertex v can be easily checked as follows. If $I < T[v]$, v is still forbidden by the tabu list. Otherwise (i.e., $I \geq T[v]$), vertex v is not forbidden by the tabu list.

3.4.3 The time complexity of CBTS

CBTS operates directly on the input graph and uses the adjacent list representation to store the graph. Given a solution (X, Y), by our implementation, the time complexity of constructing C_{expand} and $C_{plateau}$ is bounded by $O(|U \cup V|)$ since we need to calculate δ_i for each vertex $i \in C$ while $|C|$ is bounded by $|U \cup V|$. We also need to update $|N(i) \cap X|$ and $|N(i) \cap Y|$ when one vertex is moved (outside the solution or into the solution). The time complexity of these updates is bounded by $O(\Delta)$ ($\Delta = \max_{i \in U \cup V}(|N(i)|)$). Hence, the time complexity of one iteration in CBTS is bounded by $O(|U \cup V| + \Delta)$.

3.5 Reduction by the Peel procedure

Our TSGR-MBBP algorithm employs the $Peel(G, \omega)$ procedure (Algorithm 1, line 11) to recursively delete all vertices whose degrees are smaller than or equal to ω until no such vertex exists. Obviously, if the cardinality of one vertex set of the reduced bipartite graph (which is a upper bound of the maximum biclique) is less than or equal to ω (which is a lower bound), then ω must be the optimal objective value because no better solution can exist in the reduced graph (Algorithm 1, lines 19-20). Given $G = (U, V, E)$, $Peel(G, \omega)$ can be implemented by a standard breadth-first search, thus in time $O((|U \cup V| + |E|)$.

The peeling procedure is triggered each time the balanced size of the largest biclique discovered so far (lower bound) is larger than or equal to the minimum degree of the current graph. This procedure is effective on large sparse graphs but may not reduce a dense graph much. The experiments reported in Section 4 confirm that, with a high quality lower bound, large real-life bipartite graphs can be significantly reduced.

We note that the idea of *Peel* was previously used in a GRASP heuristic for detecting dense subgraphs (quasi-cliques) in massive sparse graphs [1]. We adapted this technique for solving MBBP for the first time.

3.6 Reduction by exact search

As discussed in Section 3.1, even if exact algorithms are unpractical for solving large instances, they can be used to find the optimal solution of a subgraph of reasonable size and thus reduce the graph. Moreover, since the optimal value of the subgraph is a lower bound of the initial graph, it can be used to update the current best balanced biclique, which in turn can further reduce the current graph. The exact algorithm used by TSGR-MBBP was adapted from a well-known B&B algorithm for the maximum clique problem [5] and described in Appendix A. This is an enumeration algorithm with a worst time complexity $O(2^{|U \cup V|})$ for a bipartite graph $G = (U, V, E)$. In practice, this exact algorithm is only applied to solve a subgraph with K vertices at most (K being the largest subgraph that is estimated to be solved in reasonable time by the algorithm). It is clear that the best K depends on the adopted exact algorithm and target subgraph. According to our experiments, we set K to 100 for random dense graphs and 500 for sparse real-life networks.

4 Computational assessment

To comprehensively evaluate the proposed TSGR-MBBP algorithm as well as its components, we tested our algorithm on two sets of benchmark instances including both (dense) random graphs and massive real-life networks.

4.1 Benchmark

- Random Graphs: This set of benchmark instances includes 30 randomly generated dense graphs. In each graph, the two vertex sets U and V have an equal cardinality (i.e., $|U| = |V|$) and an edge between a pair of vertices $(u, v) \in U \times V$ exists with uniform probability p ($0 < p < 1$) which defines the edge density of the graph. For our study, we used the same random graphs tested in [29] so that the performances of different algorithms can be compared. For each combination of $n \in \{250, 500\}$ and $p \in \{0.85, 0.90, 0.95\}$ ($n = |U| = |V|$), 5 instances were generated (30 in total). These instances are thus very dense and named as "G_<n>_<p>_<id>" where $id \in \{1, 2, 3, 4, 5\}$. A theoretical analysis in [7] showed that the maximum balanced size ω in random graphs locates in range $[\frac{\ln n}{\ln(1/p)}, \frac{2*\ln n}{\ln(1/p)}]$ with high probability (when n is sufficiently large).
- The Koblenz Network Collection (KONECT) [14]: The entire collection contains hundreds of networks derived from real-life applications. Though KONECT dataset was originally designed for network analysis such as in [18,26], these large bipartite networks are also suitable for testing TSGR-MBBP. We selected 25 bipartite graphs from different categories including affiliation networks, feature networks, rating networks, folksonomies, interaction networks etc. The scales of selected graphs vary from 829 + 551 vertices and 1476 edges to 1,425,813 + 4,000,150 vertices and 8,649,016 edges. These graphs are sparse in general (normally with an edge density less than 1%) and have quite different features in terms of vertex number at each side, max degree, edge distribution entropy, diameters [1]. Since these

[1] Details of these graphs are given at `http://konect.uni-koblenz.de/networks/`

graphs originate from different sources and own diverse features, they form a representative sample of real networks for our experimental studies. Note that irrelevant graph information for MBBP like multiple edges, vertex or edge weight in some graphs is ignored.

4.2 Parameter tuning and experimental protocol

The TSGR-MBBP algorithm has three parameters: L - the tabu search depth; α - the coefficient for tabu tenure required by the Constraint_Tabu_Improve() procedure (Section 3.4); K - the threshold on the number of vertices of the subgraph for graph reduction with the exact algorithm (Section 3.6).

Since the parameters L and α are independent from the reduction procedure, we tuned them on a simplified version of TSGR-MBBP without the graph reduction procedure (i.e., by disabling lines 10-20 in Algorithm 1). We used the automatic parameter configuration package iRace [16], which implements the Iterated F-Race (IFR) method. Given $L \in \{10, 100, 1000, 5000, 10000\}$, and $\alpha \in [0, 2]$, for each parameter configuration, we used a tuning budget of 500 hook-runs, each of which representing 10 independent calls of TSGR-MBBP. To avoid over-fitting, we used 6 (out of 30) challenging random graphs covering three densities (G_500_XXX_1 and G_500_XXX_2 where XXX corresponds to 0.95, 0.90, or 0.85) as training instances. The experiments suggested that the combination $(L = 1000, \alpha = 0.30)$ was a suitable configuration for random graphs. As for KONECT graphs, the training set included “actor-movie", “bookcrossing_full-rating", “dbpedia-genre", “dbpedia-team", “github", “stackexchange-stackoverflow". The final choice of parameters was $L = 100$ and $\alpha = 1.74$.

The use of two different settings for (L, α) is mainly due to the graph structures which vary much. According to our observations, for random dense graphs, a more intensified search is needed to find quality solutions. This is achieved with a large tabu search depth ($L = 1000$) and a short tabu tenure (with $\alpha = 0.30$). On the contrary, for large real-life sparse instances, the tabu search component is able to reach local optima very quickly. As a result, it is preferable to restart more frequently the tabu search component (with $L = 100$) and diversify more strongly the search process (using a larger tabu tenure with $\alpha = 1.74$).

The third parameter K indicates the largest subgraph that can be solved in reasonable time by the exact algorithm described in Section 3.6. We set $K = 100$ for random graphs and 500 for KONECT graphs. In effect, since the random graphs we tested are very dense, they cannot be reduced by the reduction procedure, implying that no connected subgraph with less than 100 vertices exists in this set of benchmarks. A very large K is not acceptable, otherwise the computing time for exact search becomes prohibitive according to our observations for random graphs. Preliminary experiments also confirmed that the time consumption was normally insignificant (less than 2 seconds) for connected subgraphs with less than 500 vertices for sparse KONECT graphs. As the vertex number is just a rough estimation of the hardness of the subgraph for our exact algorithm, we terminate the exact algorithm if it does not finish during 10 seconds. This additional cutting-off condition prevents the algorithm from spending too much effort in searching optimal solutions for some potential hard subgraphs. If the exact search stops without giving an optimal solution, the corresponding subgraph will not be removed. It is well known that a per-instance based parameter tuning can further improve the performance. However, as we show below, our algorithm with the above parameter setting performs already very well.

TSGR-MBBP was implemented in C++ and compiled with g++ v4.4.7 with optimization flag -o3 [2]. Our experiments were performed on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Linux 2.6.32. When solving the DIMACS machine benchmark procedure 'dfmax.c' [3] without compilation optimization flag, the run time on our machine is 0.40, 2.50 and 9.55 seconds for graphs r300.5, r400.5 and r500.5 respectively.

Considering the stochastic nature of TSGR-MBBP, we ran TSGR-MBBP 20 independent times to solve each instance. For each run, we set a time limit as the stopping condition (see line 4, Algorithm 1 and Section 3.2). For the random graphs of 250 vertices, the time limit was 30 seconds (cpu clock time), while for the random graphs of 500 vertices, 60 seconds were allowed. For the KONECT instances, we prolong this limitation to 360 seconds (6 minutes) since these instances are much larger than the random graphs.

4.3 Computational results

To evaluate the performance of TSGR-MBBP, we show computational results relative to three state-of-the-art MBBP approaches:

- EA/SM [29]: This is a hybrid algorithm mixing local search, structure mutation and repair-assisted restart. EA/SM is the most recent heuristic algorithm and outperforms the precedent algorithms like in [27,28]. For our comparative study, we ran 20 times the source code of EA/SM (provided by its authors) to solve each instance, each run being limited to 200,000 fitness evaluations according to [29]. We observed that to attain its best solutions,

[2] The source code of TSGR-MBBP is available at: `https://github.com/joey001/mbbp`

[3] `dfmax:ftp://dimacs.rutgers.edu/pub/dsj/clique/`

EA/SM needed a run time ranging from 42 to 50 seconds for instances of 250 vertices and 75 to 94 seconds for instances of 500 vertices (see Table 1). Consequently, the stopping condition of EA/SM can be considered to be more favorable than that used to run our algorithm (a *cutoff time* of 30 and 60 seconds for instances of 250 and 500 vertices respectively).

- IBM CPLEX: CPLEX is one of the best commercial optimization tools for solving integer linear programming models. We ran CPLEX (version 12.6.1) 2 hours (7200 seconds) on each instance with the binary linear formulation provided in Section 2.2. Obviously, the total time given to TSGR-MBBP for 20 runs (60*20 = 1200 seconds for the random instances and 360*20 = 7200 seconds for the KONECT instances) is no more than 2 hours.
- AL_Greedy [2]. This is a (fast) greedy algorithm which solves the equivalent maximum balanced independent set problem for the bipartite complement. According to [29], this algorithm performs better than its earlier version presented in [22]. Thus, we re-implemented this algorithm and used it for our comparative study. Since AL_Greedy is a deterministic heuristic, only one run was needed to solve each instance. Moreover, AL_Greedy stops once its construction procedure reaches its end. Thus, no explicit stopping condition is required.

4.3.1 Random graphs

Table 1 reports the computational results of TSGR-MBBP together with the results of the reference approaches (EA/SM, CPLEX and AL_Greedy) on the 30 random dense graphs. Column “instance” shows the name of each instance. Column “BKV” presents the best known values reported in [29]. For TSGR-MBBP and EA/SM [4], column “best(ave)" indicates respectively the largest biclique size among the 20 best balanced bicliques found in 20 runs, and the average size of these 20 best balanced bicliques between parentheses. Column “time” indicates the total runtime (in seconds) per run and per instance. For TSGR-MBBP, column “h-time” additionally reports the average time (in seconds) of the 20 timestamps of first hitting the best balanced sizes of 20 runs (notice that the code of EA/SM does not provide information of first hitting its best solution). For CPLEX, we report the best lower bounds and the time needed to complete the search. If CPLEX fails to report a feasible solution for an instance due to memory limitation, “-" is used in the corresponding entries of columns “best” and “time”. For the deterministic heuristic AL_Greedy, since its run time is negligible (shorter than 0.2 second for all instances), we only report the best biclique values.

[4] The results obtained by running the code of EA/SM on our computer (column “EA/SM") are slightly different from the results initially reported in [29] (column “BKV"). This can be explained by the stochastic nature of the EA/SM algorithm.

Table 1
Computational results of TSGR-MBBP together with the results of EA/SM [29], CPLEX (version 12.6.1) and AL_Greedy [2] on the set of 30 random dense graphs.

instance	BKV [29]	TSGR-MBBP			EA/SM		CPLEX 12.6.1		AL_Greedy
		best(ave)	h-time	time	best(ave)	time	best	time	best
G_250_0.95_1	68	68(68)	0.05	30	68(67.90)	50.28	66	≥7200	64
G_250_0.95_2	66	66(66)	0.21	30	66(65.05)	49.31	64	≥7200	59
G_250_0.95_3	70	70(70)	0.17	30	70(69.50)	48.87	-	-	67
G_250_0.95_4	68	68(68)	0.42	30	68(67.10)	47.36	66	≥7200	63
G_250_0.95_5	68	68(68)	0.72	30	67(66.95)	47.41	67	≥7200	62
G_250_0.90_1	44	44(44)	0.06	30	44(43.70)	42.94	42	≥7200	37
G_250_0.90_2	44	**45**(45)	0.52	30	45(43.90)	43.28	42	≥7200	39
G_250_0.90_3	44	44(44)	0.13	30	44(43.45)	43.20	42	≥ 7200	40
G_250_0.90_4	45	45(45)	0.66	30	44(43.80)	43.13	42	≥ 7200	40
G_250_0.90_5	45	45(45)	0.23	30	45(44.10)	45.13	41	≥7200	40
G_250_0.85_1	33	33(45)	0.11	30	33(32.40)	47.92	-	-	30
G_250_0.85_2	33	33(33)	0.04	30	33(32.75)	49.94	-	-	31
G_250_0.85_3	34	34(34)	0.69	30	34(32.95)	44.66	-	-	31
G_250_0.85_4	33	33(33)	0.07	30	33(32.90)	43.76	30	≥7200	30
G_250_0.85_5	33	33(33)	0.52	30	33(32.30)	44.16	30	≥7200	30
G_500_0.95_1	91	**93**(93)	14.37	60	91(90.20)	93.28	-	-	83
G_500_0.95_2	89	**91**(91)	15.58	60	90(88.30)	92.02	-	-	81
G_500_0.95_3	89	**91**(90.05)	3.85	60	90(87.85)	92.62	85	≥ 7200	81
G_500_0.95_4	88	**90**(89.40)	21.04	60	88(86.85)	93.28	83	≥ 7200	78
G_500_0.95_5	90	**91**(90.90)	13.40	60	90(88.15)	94.30	81	≥ 7200	83
G_500_0.90_1	56	56(56)	12.21	60	55(53.75)	76.24	46	≥ 7200	49
G_500_0.90_2	56	56(56)	5.38	60	56(54.00)	79.34	47	≥ 7200	48
G_500_0.90_3	54	**56**(55.60)	15.57	60	55(53.45)	79.52	46	≥ 7200	48
G_500_0.90_4	55	**56**(55.55)	9.87	60	55(53.75)	79.59	47	≥ 7200	48
G_500_0.90_5	55	**56**(55.50)	13.68	60	55(53.25)	82.23	44	≥ 7200	48
G_500_0.85_1	40	40(40)	4.59	60	40(38.45)	75.55	33	≥ 7200	34
G_500_0.85_2	41	41(41)	5.84	60	40(39.25)	75.56	32	≥7200	33
G_500_0.85_3	40	**41**(40.50)	13.50	60	41(38.65)	81.48	35	≥7200	35
G_500_0.85_4	40	40(40)	1.84	60	39(38.30)	75.29	33	≥7200	35
G_500_0.85_5	41	41(41)	4.60	60	40(38.60)	75.06	31	≥7200	34

From Table 1, we first observe that in terms of solution quality, TSGR-MBBP competes very favorably with the reference approaches. In particular, TSGR-MBBP improves the best-known results reported in [29] for 10 instances (marked in bold font). For the 20 remaining instances, the best objective values found by TSGR-MBBP are always as good as or better than those of the reference algorithms. When The average objective values of the 20 runs of TSGR-MBBP are also better than that of EA/SM. Moreover, the performance of TSGR-MBBP is quite stable across the whole set of tested instances. In terms of computational efficiency, TSGR-MBBP is also very competitive. It finds the solutions within no more than 30 and 60 seconds for the instances of

250 and 500 vertices, in contrast to the reference algorithm EA/SM that finds equal or inferior solutions in around 45 and 90 seconds respectively. As for CPLEX, it cannot complete its search within a duration of 2 hours and thus fails to find the optimal solution for any instance (still CPLEX finds some solutions better than those of AL_Greedy). Unsurprisingly, the greedy algorithm AL_Greedy leads to solutions of very poor quality. We also observed that no vertex can be reduced for these random instances, indicating that the degrees of all vertices are larger than the best balanced size. In fact, this is expected according to [7] where an estimation of the size of the maximum balanced biclique for random bipartite graphs is presented. For the graphs of Table 1, the maximum balanced biclique is estimated to be much smaller than the average vertex degree. For instance, the vertex degree of “G_500_0.85_X” is closely around 425 while the maximum balanced biclique is estimated to have a size between 39 and 76. However, as we show in the next section, the reduction procedure becomes extremely effective when large sparse graphs are considered.

Finally, to verify whether the observed performance differences are statistically significant, we apply the non-parametric Friedman test to compare TSGR-MBBP against EA/SM and AL_Greedy in terms of the best and average objective values. For the best results, the p-value of 0.0106 from the test between the results of TSGR-MBBP and EA/SM indicates a significant difference. The test between TSGR-MBBP and AL_Greedy leads to a p-value even smaller than 0.00001, indicating there is a significant different between the compared results. As to the average results (only applicable to TSGR-MBBP vs EA/SM), the p-value smaller than 0.00001 again confirms that the difference is significant. This study demonstrates that our algorithm competes very favorably with the three reference approaches on this set of benchmark instances.

4.3.2 KONECT networks

We report in Table 2 the computational results of TSGR-MBBP and CPLEX on the set of 25 KONECT instances. For this study, we ignore the results of EA/SM and AL_Greedy. The code of EA/SM cannot be run with these graphs (it cannot load a graph with more than 30,000 vertices on each side). AL_Greedy runs out of memory for large graphs with more than 30,000 vertices. For small instances such as moreno_crime and opsah-ucforum, Al_Greedy returns a trivial biclique of size zero. Columns “name”, “$(|U|, |V|)$”, “$|E|$” show the basic information of the original instances. Columns “best(ave)", “h-time" (for TSGR-MBBP only) and ‘time" report the same information as in Table 1. For TSGR-MBBP, columns “red_1” and “red_2” indicate the average number of reduced vertices in all runs by the first and second reduction rules respectively. To enable CPLEX to load large graphs, each original graph

Table 2
Computational results of TSGR-MBBP and CPLEX on the set of 25 large KONECT instances. The results of EA/SM and AL_Greedy are not available.

instance			TSGR-MBBP					CPLEX		
name	$(\|U\|, \|V\|)$	$\|E\|$	best (ave)	h-time	time	red_1	red_2	$(\|U'\|, \|V'\|)$	best	time
actor-movie	(127823, 383640)	1470418	8(8)	8.91	360	474872.4	306.6	(100398, 88729)	N/A	N/A
bibsonomy-2ui	(5794, 767447)	2555080	8*(8)	1.01	360	772062.0	1179.0	(137, 307)	8*	2209.76
bookcrossing_full-rating	(105278, 340523)	1149739	13(12.30)	122.25	360	432401.7	19.8	(26799, 76949)	N/A	N/A
dblp-author	(1425813, 4000150)	8649016	10*(10)	8.92	360	5400509.4	25453.6	(0, 0)	-	-
dbpedia-genre	(258934, 7783)	463497	7*(7)	1.22	360	265993.1	723.9	(385, 118)	7*	931.59
dbpedia-location	(172091, 53407)	293697	5*(5)	0.22	360	224672.2	825.8	(0, 0)	-	-
dbpedia-occupation	(127577, 101730)	250945	6*(6)	0.88	360	228851.0	456.0	(0, 0)	-	-
dbpedia-producer	(48833, 138844)	207268	6*(6)	0.17	360	183879.0	3798.0	(0, 0)	-	-
dbpedia-recordlabel	(168337, 18421)	233286	6*(6)	0.23	360	186512.8	245.2	(0, 0)	-	-
dbpedia-starring	(76099, 81085)	281396	6*(6)	0.29	360	156444.0	740.0	(44, 21)	6*	2.06
dbpedia-team	(901166, 34461)	1366466	6(5.50)	99.29	360	898778.3	309.2	(24858, 4345)	N/A	N/A
dbpedia-writer	(89356, 46213)	144340	6*(6)	0.13	360	131812.9	3756.1	(0, 0)	-	-
discogs_affiliation	(1754823, 270771)	14414659	26(26)	22.15	360	2009185.1	379.9	(11722, 4307)	N/A	N/A
discogs_lgenre	(270771, 15)	4147665	15*(15)	10.16	360	270726.3	59.7	(0, 0)	-	-
discogs_style	(1617943, 383)	24085580	38(37.15)	131.85	360	1612358.4	0.0	(5289, 305)	36	$\geq$ 7200
edit-frwiki	(288275, 4022276)	46168355	41(27.50)	228.91	360	4162813.8	1.6	(6664, 56700)	N/A	N/A
edit-frwiktionary	(5017, 1907247)	7399298	19(19)	31.88	360	1909273.0	0.0	(232, 2759)	16	$\geq$ 7200
flickr-groupmemberships	(395979, 103631)	8545307	67(67)	94.74	360	457907.3	145.7	(213863, 61790)	N/A	N/A
github	(56519, 120867)	440237	12(12)	4.74	360	169911.1	637.9	(4001, 2836)	N/A	N/A
moreno_crime	(829, 551)	1476	2*(2)	0.00	360	1147.0	233.0	(4, 4)	2*	0.03
opsahl-collaboration	(16726, 22015)	58595	8*(8)	0.05	360	37416.3	1324.7	(0, 0)	-	-
opsahl-ucforum	(899, 522)	33720	5*(5)	0.03	360	560.7	860.3	(0, 0)	-	-
stackexchange-stackoverflow	(545196, 96680)	1301942	9(8.95)	92.12	360	625182.8	49.7	(1432, 867)	8	$\geq$ 7200
wiki-en-cat	(1853493, 182947)	3795796	14*(14)	17.58	360	2028030.4	8409.5	(87, 60)	14*	11.23
youtube-groupmemberships	(94238, 30087)	293360	12(12)	0.81	360	121958.3	67.7	(1432, 867)	8	$\geq$ 7200

was pre-reduced by applying $Peel(G, best)$ before starting CPLEX. Column "$(|U'|, |V'|)$" reports the number of vertices after applying $Peel(G, best)$ while columns "best" and "time" report the best balanced size reached as well as the total consumed time. Symbol "*" indicates that the solution has been proven to be optimal by the corresponding algorithm, while symbol "-" means that the initial (and *Peel* pre-reduced) graph cannot be loaded into CPLEX.

As explained in Section 3.2, when either of the two vertex sets of the current bipartite graph contains less than ω (the best balanced size found so far) vertices, ω is proven to be the optimal maximum balanced size. From Table 2, we observe that TSGR-MBBP proves optimality for 14 out of the 25 instances (indicated by "*"), even though these real-world instances are significantly larger than the random instances. Also, TSGR-MBBP achieves the same best balanced size in all 20 runs for all but 5 instances (whose average objective values are reported in the table). Observing the number of vertices that has been reduced, we find that the first reduction method (the *Peel* method) prunes more than half or even all of the vertices during the search procedure. As for the second reduction method (which is based on exact search), though the vertices removed by this method are fewer than the first method, we cannot neglect its significance. For 5 instances "bibsonomy-2ui","dpedia-genre","dbpedia-starring", "moreno_crime" and "wiki-en-cat", the *Peel* procedure fails to reduce these graphs to small enough subgraphs such that optimality can be proven (one vertex set of the subgraph includes fewer than ω vertices, see column "$(|U'|, |V'|)$"), TSGR-MBBP directly finds the optimal solution for the resulting subgraphs with less than K vertices. The CPLEX solver, unfortunately, is unable to load some of these massive graphs even after reducing these graphs significantly by applying $Peel(G, best)$ in the pre-processing step. For instances for which CPLEX finds a feasible solution, like "discogs_style", "edit-frwiktionary", "stackexchange-stackoverflow" and "youtube-groupmemberships", the results are still worse than those achieved by TSGR-MBBP. Besides, CPLEX always requires a longer time than TSGR-MBBP to attain the best solution.

4.4 Comments on the reduction methods

As shown in Section 4.3.1, graph reductions are unsuccessful for dense random graphs (no vertex has been reduced for the tested instances). In other words, the results of our TSGR-MBBP algorithm for these dense graphs have been achieved only with its constraint-based tabu search component CBTS described in Section 3.4. This indicates that the tabu search procedure alone competes very favorably compared to the existing algorithms for these random dense graphs.

Now one interesting question is when and why graph reductions can contribute positively to the performance of TSGR-MBBP. As shown in Table 2 of Section 4.3.2, graph reductions perform very well for large sparse graphs. Generally, when there exists a high number of vertices with small degrees in the given graph, *Peel* can effectively remove these vertices and thus reduce the search space. This kind of graphs is often characterized by low densities or small degeneracies [23]. Moreover, if *Peel* separates the current graph into some small subgraphs, the second reduction could further reduce the search space and improve the current lower bound. We observed that these graphs are often composed by communities with different sizes [12]. Indeed, the above two characteristics are very common in graphs from real-life applications, which explain why graph reductions work so well on the tested KONECT instances.

Finally, note that the graph reduction techniques are general and independent from the tabu search component. As a result, these techniques (and other possible reductions) can be advantageously employed in combination with other search algorithms like EA/SM [29] in order to obtain improved performances.

5 Analysis of CBTS and reduction methods

This section presents an empirical analysis of the restricted unbalance constraint related to the Constraint-Based Tabu Search procedure (Section 3.4) and the merit of the graph reduction procedure (Section 3.5).

5.1 Unbalance constraint of Constraint-Based Tabu Search

The Constraint-Based Tabu Search procedure (see Sections 3.4) is one key component of the proposed TSGR-MBBP algorithm. One of the main features of CBTS is that while unbalanced bicliques are allowed, the balance deviation of the explored bicliques cannot exceed 2 (this constraint is called unbalance limit). To justify this specific unbalance limit, we compare CBTS with two CBTS versions with different unbalance limits. The first version (called “$\text{CBTS}_{\Omega^\infty}$") removes the unbalance limit and allows the procedure to visit bicliques with any unbalance (lines 17-23 are removed from Algorithm 2). The second version (named as “CBTS_{Ω^1}") introduces a more restrictive unbalance limit – the balance deviation is required to be no more than 1 after each iteration (i.e., change the repairing condition in line 17 of Algorithm 2 to $|X| - |Y| > 1$). We also used “CBTS_{Ω^2}" to denote the original CBTS procedure. As such, these three CBTS versions correspond to three restart algorithms searching within the solution spaces Ω^2, Ω^∞, and Ω^1 respectively. Note that the version with absolute balanced constraint is not considered. In

Table 3
Comparison between three different versions of the Constraint-Based Tabu Search procedure: CBTS$_{\Omega^\infty}$ can visit any biclique; CBTS$_{\Omega^1}$ visits only bicliques with a balance deviation no more than 1; CBTS$_{\Omega^2}$ (the original version of CBTS) visits bicliques with a balance deviation no more than 2.

instance	CBTS$_{\Omega^\infty}$			CBTS$_{\Omega^1}$			CBTS$_{\Omega^2}$		
	best(ave)	h-time	iter	best(ave)	h-time	iter	best(ave)	h-time	iter
GraphU_500_0.05_3	90(89.20)	8.48	2827917	70(68.25)	19.63	1387704	91(90.20)	7.76	2297846
GraphU_500_0.05_4	90(88.20)	19.48	2886301	69(67.35)	19.47	1386318	90(89.20)	10.99	2288732
GraphU_500_0.05_5	91(89.95)	20.37	2829679	71(68.40)	28.71	1389515	91(90.95)	16.41	2287010
GraphU_500_0.10_3	54(53.85)	18.77	5620994	42(40.70)	15.32	1387011	56(55.60)	13.49	2343998
GraphU_500_0.10_4	55(54.20)	17.11	5774901	42(40.90)	16.74	1385316	56(55.30)	7.17	2356738
GraphU_500_0.10_5	55(54.10)	10.04	5750204	42(41.45)	22.12	1388886	56(55.55)	13.27	2352404
GraphU_500_0.15_3	39(38.55)	15.57	6340722	31(29.75)	18.62	1441000	41(40.70)	16.92	2409855
dblp-author	8(5.40)	26.69	2674721	10(8.60)	27.25	1089614	10(9.50)	21.27	696636
dbpedia-genre	5(2.85)	18.07	546103	4(2.85)	19.69	121851	4(3.05)	9.35	147990
dbpedia-team	4(3.25)	16.25	5699436	4(3.30)	8.94	1232615	5(3.85)	24.11	1260575
discogs_style	9(3.30)	1.19	10651	7(3.80)	5.14	13617	25(7.20)	29.43	19900
edit-frwiktionary	9(2.65)	0.18	2476	9(2.85)	5.52	2204	9(3.05)	1.28	1946
wiki-en-cat	14(6.05)	29.28	3640199	13(7.50)	24.17	553078	14(8.75)	25.18	706691

effect, if we repair the solution whenever $|X| - |Y| \neq 0$, the current solution can never be improved because the "push" operator only imports one vertex to one vertex set of the current biclique in each iteration.

For this study, we used 13 instances selected from the two benchmark sets. We ran each CBTS version 20 trials to solve each instance using the protocol of Section 4.2. Each trial was given a time limit of 60 seconds. The comparative results are summarized in Table 3. We denote one restart of CBTS as one iteration here (one "while" loop, lines 10-20 in Algorithm 1). Column "best(ave)" indicates the best biclique size over 20 runs and the average of the 20 best biclique sizes found during 20 runs. "h-time" reports the average of 20 timestamps of first hitting the best bicliques of 20 runs. Column "iter" reports the average number of restarts over 20 runs.

As for the solution quality, the original Constraint-Based Tabu Search (CBTS$_{\Omega^2}$) procedure dominates the other variants both in terms of best and average values. CBTS$_{\Omega^2}$ also performs the best concerning the average time of hitting the best solution for random graphs. As for the total number of iterations (column "iter"), CBTS$_{\Omega^\infty}$ restarts more often than CBTS$_{\Omega^2}$ which on the other hand restarts more often than CBTS$_{\Omega^1}$. Obviously, a tighter unbalance limit leads to more frequent calls to the repair procedure, thus less iterations under the same cutoff time limit. Meanwhile, the results suggest that the strategy of incorporating unbalance constraint is a good trade-off between solution quality and number of iterations.

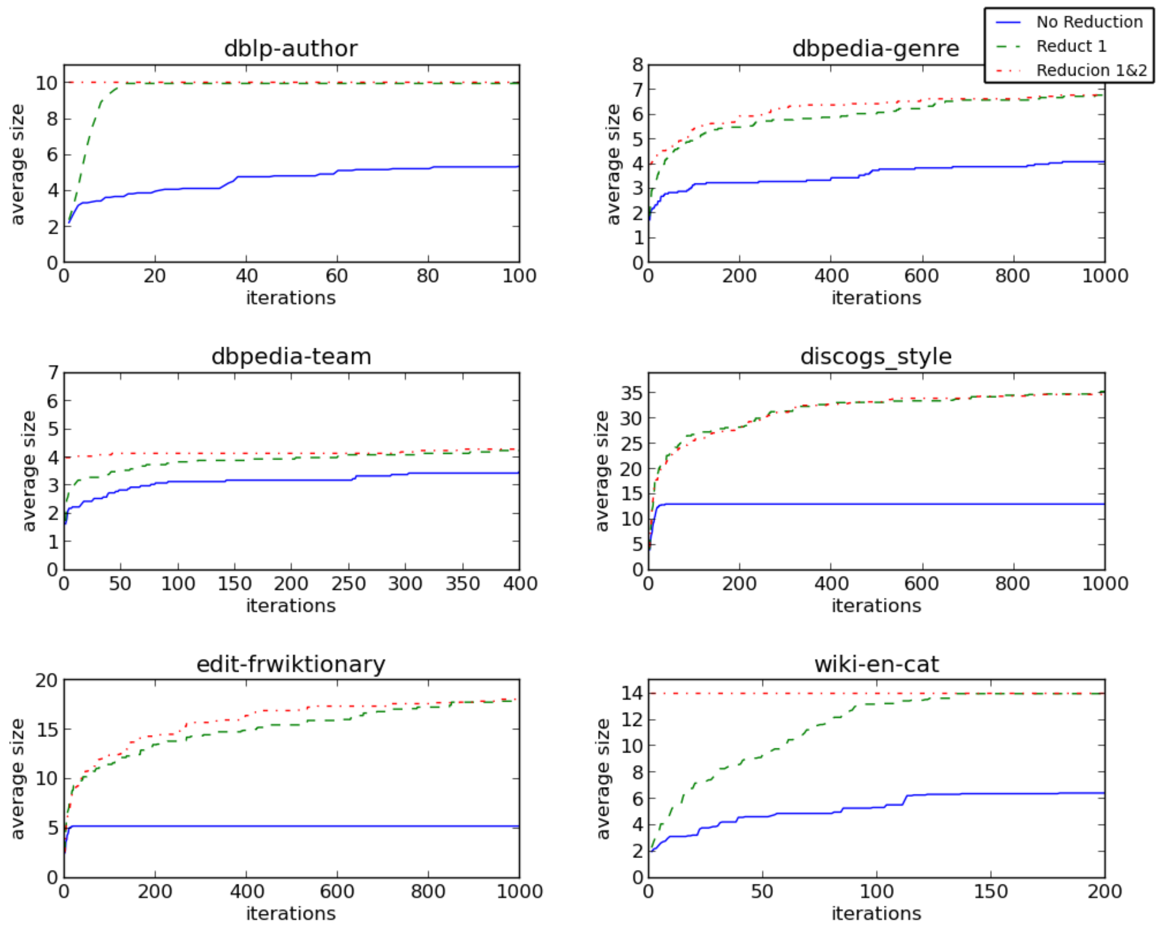

Fig. 3. The relations between the number of iterations and the average best sizes of 20 runs on 6 selected instances from KONECT.

5.2 Effectiveness of reduction methods

To gain a comprehensive understanding of the run-time behavior and efficiency of the two reduction methods, we show in this section an analysis of the convergence rate of three variants of the TSGR-MBBP algorithm:

- *No Reduction:* The reduction procedure is disabled, i.e., lines 10-18 are removed from Algorithm 1.
- *Reduction 1:* Only the first reduction method (the *Peel* method) is used. i.e., lines 12-18 are removed from Algorithm 1.
- *Reduction 1&2:* Both reduction methods are used, i.e., the original TSGR-MBBP algorithm.

The variant with only the second reduction is not considered as the exact search will never be triggered without the *Peel* procedure.

This study was based on 6 KONECT instances, “dblp-author", “dbpedia-genre", “dbpedia-team", “discog_style", “edit-frwikitionary" and “wiki-en-cat" which are large enough with different levels of difficulty for TSGR-MBBP (the difficulty is estimated by the time consumption of TSGR-MBBP in Table 2). We ran each algorithm variant 20 times to solve each instance with a time limit of 6 minutes per run. Again, we denote one restart of CBTS as one iteration. Figure 3 reports the relation between the number of iterations and the average best balanced size reached by each variant in 20 runs (abbreviated as ‘average size’). Considering the two variants with reduction can stop before reaching the time limit when the optimum is proven, we assume that the best size after termination is constantly the optimal size in this case.

According to Figure 3, in terms of the average result after the same number of iterations, the two variants using reduction always dominate the variant without reduction. Actually, “No Reduction" converges so slowly that it even has difficulties in reaching half of the best-known size in the given time limit. Comparing “Reduction 1" and “Reduction 1&2", for “dblp-author", “dbpedia-genre", “dbpedia-team", “edit_fiwikitionaryand" and “wiki-en-cat", “Reduction 1&2" always discovers solutions of high quality earlier. In particular, for two instances, “dblp-author" and “wiki-en-cat", “Reduction 1&2" reaches the optimal solution in the very first iteration. This is because for these graphs, the exact algorithm discovered the optimal solution in some of the connected subgraphs at the beginning of the search, which in turn enabled the *peel* procedure to prune the graph to trivial size and thus proves the global optimality. Nevertheless, for “discogs_style", “Reduction 1&2" and “Reduction 1" perform similarly. We also notice that the curves of “Reduction 1" and “Reduction 1&2" meet sooner or later for all the instances. In a nutshell, the convergence rate is highly related to the instance under consideration, but in any case, both reduction methods accelerate the search procedure.

6 Conclusions and perspectives

The Maximum Balanced Biclique Problem is of great interest both theoretically and practically. We have presented an original algorithm combining tabu search and two graph reduction techniques for solving MBBP. The proposed TSGR-MBBP algorithm is driven by its Constraint-Based Tabu Search (CBTS) procedure to retrieve high quality solutions from the current graph. CBTS employs the “push" operator to explore a relaxed search space including both balanced and unbalanced bicliques while imposing a specific unbalance limit on explored solutions. Moreover, This search process is coupled with two reduction rules to prune the graph, which leads to a reduced search space for the following iterations.

The proposed algorithm has been assessed on two benchmark sets: 30 random dense instances and 25 real-life large instances from the KONECT collection. For the random instances, TSGR-MBBP dominates existing state-of-the-art approaches EA/SM [29], GL_Greedy [2] and CPLEX (version 12.6.1). Besides, new improved solutions (new lower bounds) were found by TSGR-MBBP for 10 out of the 30 instances. For the KONECT instances, TSGR-MBBP proved optimal solutions for 14 instances and found high quality solutions for the other instances. Experiments have also indicated that TSGR-MBBP performs better than CPLEX both in terms of solution quality and computation time. Besides, we have also noticed that the two reduction methods are able to prune a significant number of vertices for large sparse graphs. Given that graph reductions are general and independent from the search procedure, they can be advantageously used in combination of other search algorithms.

This study can be extended in several directions. First, it would be interesting to study other customized moves based on the “push" operator. Second, this work showed that graph reductions can significantly prune large sparse graphs from real-world applications. It is then interesting to investigate other reduction techniques to further simplify the given graph. Third, as shown in our literature review, there are few exact algorithms for MBBP. It would be useful to design more elaborated exact algorithms, for instance, by adapting advanced exact clique algorithms like [20,21] to MBBP. Finally, given the effectiveness of the reduction techniques on large-scale instances for MBBP, it would be interesting to investigate similar techniques in the context of other clique related problems like the maximum clique problem [25] and the maximum k-plex problem [4,30] for massive graphs.

Acknowledgment

We are grateful to the anonymous referees for valuable suggestions and comments which helped us to improve the paper. We’d like to thank the authors of [29] for sharing the code of their EA/SM algorithm and the random instances used in this work. Support for the first author from the China Scholarship Council (2013-2017) is also acknowledged.

References

[1] J. Abello, M.G. Resende, S. Sudarsky, Massive quasi-clique detection, in: Latin American Symposium on Theoretical Informatics, Springer, 2002, Lecture Notes in Computer Science 2286, pp. 598–612.

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

[3] 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.

[4] B. Balasundaram, S. Butenko, I.V. Hicks, Clique relaxations in social network analysis: The maximum k-plex problem, Operations Research 59 (1) (2011) 133–142.

[5] R. Carraghan, P.M. Pardalos, An exact algorithm for the maximum clique problem, Operations Research Letters 9 (6) (1990) 375–382.

[6] Y. Cheng, G.M. Church, Biclustering of expression data., in: Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, San Diego, California, USA. August 19-23, 2000, pp. 93–103.

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

[8] U. Feige, S. Kogan, Hardness of approximation of the balanced complete bipartite subgraph problem, Tech. rep., Weizmann Inst. Sci (2004).

[9] B. Galitsky, Improving relevance in a content pipeline via syntactic generalization, Engineering Applications of Artificial Intelligence 58 (2017) 1–26.

[10] 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.

[11] A. Giovannucci, J.A. Rodríguez-Aguilar, A. Reyes, F.X. Noria, J. Cerquides, Enacting agent-based services for automated procurement, Engineering Applications of Artificial Intelligence 21 (2) (2008) 183–199.

[12] M. Girvan and M. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences, 99 (12) (2002) 7821–7826.

[13] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

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

[15] J. Li, K. Sim, G. Liu, L. Wong, Maximal quasi-bicliques with balanced noise tolerance: concepts and co-clustering applications, in: Proceedings of the 2008 SIAM International Conference on Data Mining, SIAM, 2008, pp.72–83.

[16] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[17] C. McCreesh, P. McCreesh, An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem, in: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, 2014, Lecture Notes in Computer Science 8451, pp. 226–234.

[18] R. Mohamadi-Baghmolaei, N. Mozafari , H. Ali, Trust based latency aware influence maximization in social networks, Engineering Applications of Artificial Intelligence 41 (2015) 195–206.

[19] D. Mubayi, G. Turán, Finding bipartite subgraphs efficiently, Information Processing Letters 110 (5) (2010) 174–177.

[20] B. Pattabiraman, M. Ali Patwary, A.H. Gebremedhin, W. Liao, A. Choudhary, Fast algorithms for the maximum clique problem on massive sparse graphs, in: Proceedings of International Workshop on Algorithms and Models for the Web-Graph, Springer, 2013, Lecture Notes in Computer Science 8305, pp. 156–169.

[21] P. San Segundo, A. Lopez, P. M. Pardalos, A new exact maximum clique algorithm for large and massive sparse graphs, Computers & Operations Research 66 (2016) 81–94.

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

[23] A. Verma, A. Buchanan S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks, INFORMS Journal on Computing, 27 (1) (2015) 164-177.

[24] Q. Wu, J.K. Hao, Solving the winner determination problem via a weighted maximum clique heuristic, Expert Systems with Applications, 42 (1) (2015) 355–365.

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

[26] A.M. Yagci, T. Aytekin, F.S, Gurgen, Scalable and adaptive collaborative filtering by mining frequent item co-occurrences in a user feedback stream, Engineering Applications of Artificial Intelligence 58 (2017) 171–184.

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

[28] 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.

[29] 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.

[30] Y. Zhou, J.K. Hao, Frequency-driven tabu search for the maximum s-plex problem, Computers & Operations Research 86 (2017) 65–78.

[31] 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.

[32] Y. Zhou, A. Rossi, J.K. Hao, Towards effective exact algorithms for the maximum balanced biclique problem in bipartite graphs, European Journal of Operational Research 269 (3) (2018) 834–843.

A The exact search algorithm

The exact algorithm (Algorithm 3 and Proc. bbexpand) used in TSGR-MBBP is adapted from the classical B&B algorithm for the maximum clique problem [5]. Instead of starting from the trivial lower bound 0, our exact algorithm receives an initial lower bound ω (Algorithm 3, line 2), which is the best balanced size ever found in TSGR-MBBP. (X^*, Y^*), the best biclique found by the exact algorithm, is initialized as a tuple of two empty sets (Algorithm 3, line 3). If there is no solution better than ω in the current subgraph, (X^*, Y^*) remains empty even after the exact search. In such a case, the real optimal solution is discarded as we are only interested in solutions better than ω. The exact algorithm calls a recursive procedure *bbexpand* (Proc. bbexpand) to start the branch and bound search.

Unlike the original algorithm in [5], which only builds one set that forms a clique, the *bbexpand* procedure alternatively builds two sets A and B ($|A| = |B|$ or $|A| + 1 = |B|$) such that A and B form a biclique. Sets C_A and C_B contain the candidate vertices that may be added to A and B respectively, i.e., ($C_A = \bigcap_{i \in B} N(i)$ and $C_B = \bigcap_{i \in A} N(i)$). Each invocation of *bbexpand* recursively traversals the feasible bicliques containing A and B with all possible combinations of C_A and C_B examined. The procedure works as follow:

Firstly, if candidate set C_A is empty, *bbexpand* tries to update the current lower bound lb (lines 1-5, Proc. bbexpand). As the cardinality of set A is equal to or one less than that of B in the input of *bbexpand*, $|A|$ is always the balanced size of biclique (A, B) (or (B, A)). Then, if C_A is not empty, *bbexpand* enters a while loop (lines 6-14), where in each iteration, a vertex i with minimum index is picked from C_A (lines 9-10) to form a new solution with $A \cup \{i\}$ and B while removing i from C_A at the same time. In the end of the iteration, *bbexpand* is recursively called to enumerate all the feasible bicliques with the new sets of solution and new candidate sets (C_A and $C_B \cap N(i)$). Note that the roles of A and B as well as C_A and C_B in the next level of recursive call (lines 11-14) are exchanged. This is because that, to meet the balance constraint, the biclique is built by alternatively introducing a vertex from U and V. The if part in lines 7-8 prunes the unnecessary expanding since, when $|A| + |C_A|$ is smaller than the current lower bound, there is no possibility to discover a better solution based on the given solution and candidate sets. This simple rule of pruning unnecessary enumeration is similar to the one used in [5]. In brief, every loop from line 6 to line 14 enumerates possible bicliques involving set A with a newly selected vertex i and set B.

Algorithm 3: Exact search algorithm

```
Input: Graph instance $G(U, V, E)$, initial lower bound $\omega$
Output: A biclique $(X^*, Y^*)$ with maximum balanced size
1  begin
2      $lb \leftarrow \omega$ ;                     /* Initialize the lower bound as $\omega$ */
3      $(X^*, Y^*) \leftarrow (\emptyset, \emptyset)$ ;   /* Initialize the best solution as empty sets */
4      bbexpand($G$, $\emptyset$, $\emptyset$, $U$, $V$);
5  return $(X^*, Y^*)$
```

Procedure bbexpand(G, A, B, C_A, C_B)

```
Input: Graph instance $G = (U, V, E)$, $A$, $B$ - current sets that forms a biclique,
       $C_A$, $C_B$ - the sets of eligible vertices that can be added to $A$ and $B$
       respectively.
Output: The maximum balanced size $\omega$ in $G$, the biclique $(X^*, Y^*)$ with balanced
        size $\omega$.
1  if $|C_A| = 0$ then
2      if $|A| > \omega$ then
3          $lb \leftarrow |A|$ ;
4          Record current solution $(A, B)$ in $(X^*, Y^*)$;
5          return
6  while $C_A \neq \emptyset$ do
7      if $|A| + |C_A| \leq lb$ then
8          return
9      $i \leftarrow min\{i | i \in C_A\}$;
10     $C_A \leftarrow C_A \setminus \{i\}$;
11     if $|A| < |B|$ then
12         bbexpand($G$,$A \cup \{i\}$, $B$, $C_A$, $C_B \cap N(i)$)
13     else
14         bbexpand($G$,$B$, $A \cup \{i\}$, $C_B \cap N(i)$, $C_A$)
15 return
```