Finding Balanced Bicliques in Bipartite Graphs Using Variable Neighborhood Search

Juan David Quintana, Jesús Sánchez-Oro(✉), and Abraham Duarte

Department of Computer Sciences, Universidad Rey Juan Carlos, Móstoles, Spain
{juandavid.quintana,jesus.sanchezoro,abraham.duarte}@urjc.es

Abstract. The Maximum Balanced Biclique Problem (MBBP) consists of identifying a complete bipartite graph, or biclique, of maximum size within an input bipartite graph. This combinatorial optimization problem is solvable in polynomial time when the balance constraint is removed. However, it becomes $\mathcal{NP}$–hard when the induced subgraph is required to have the same number of vertices in each layer. Biclique graphs have been proven to be useful in several real-life applications, most of them in the field of biology, and the MBBP in particular can be applied in the design of programmable logic arrays or nanoelectronic systems. Most of the approaches found in literature for this problem are heuristic algorithms based on the idea of removing vertices from the input graph until a feasible solution is obtained; and more recently in the state of the art an evolutionary algorithm (MA/SM) has been proposed. As stated in previous works it is difficult to propose an effective local search method for this problem. Therefore, we propose the use of Reduced Variable Neighborhood Search (RVNS). This methodology is based on a random exploration of the considered neighborhoods and it does not require a local search.

Keywords: Biclique · Reduced VNS · Bipartite

1 Introduction

Let $G(L, R, E)$ be a balanced bipartite graph where L and R are the two sets (or layers) of vertices of the same cardinality (i.e., $|L| = |R| = n$) and E is the set of edges. As a bipartite graph, $L \cap R = \emptyset$, and an edge can only connect a vertex $v \in L$ with a vertex $u \in R$, i.e., $\forall (v, u) \in E,\ v \in L \wedge u \in R$. Additionally, let us define a biclique $B(L'R', E')$ as an induced subgraph of G, where $L' \subseteq L$, $R' \subseteq R$, such as every vertex $v \in L'$ is connected to all the vertices $u \in R'$. In other words, B is a complete bipartite graph.

Given a balanced bipartite graph $G(L, R, E)$, this work is focused on solving the Maximum Balanced Biclique Problem (MBBP), which consists of identifying

This work has been partially founded by Ministerio de Economía y Competitividad with grant ref. TIN2015-65460-C2-2-P.

a balanced biclique $B^\star(L', R', E')$ with the largest number of vertices per layer. In other words, the objective of MBBP is maximizing the cardinality of sets L' and R'.

Figure 1 presents an example bipartite graph with 8 vertices and 13 edges and two possible solutions for the MBBP. Figure 1(b) depicts a solution $B_1(L_1, R_1, E_1)$ with two vertices in each layer. Specifically, $L_1 = \{\mathtt{A}, \mathtt{B}\}$, and $R_1 = \{\mathtt{F}, \mathtt{G}\}$. The edges involved in the induced biclique are depicted with continuous line, while those with an endpoint out of the solution are depicted with dashed line. Notice that it is not possible to insert new vertices in the solution, since the resulting induced bipartite graph will not be a balanced biclique. For instance, adding vertices E or H is not possible because they are not adjacent to vertices B and A, respectively. Furthermore, it is not possible to add new vertices just in layer L_1 since the induced biclique is not balanced (i.e., $|L_1| \neq |R_1|$).

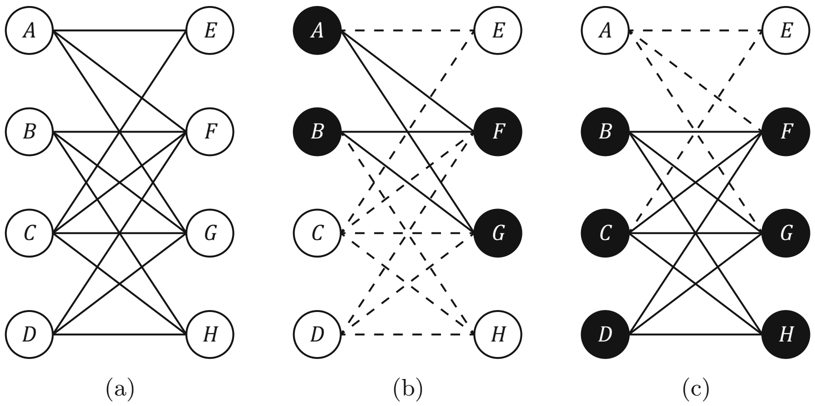

Fig. 1. (a) Bipartite graph with 8 vertices and 13 edges, (b) a feasible solution with 2 vertices in each layer (A and B in L_1, F and G in R_1), and (c) a different solution with 3 vertices in each layer (B, C and D in L_2, F, G, and H in R_2)

Figure 1(c) presents a solution $B_2(L_2, R_2, E_2)$ of better quality, since it has 3 vertices in each layer. In particular, $L_2 = \{\mathtt{B}, \mathtt{C}, \mathtt{D}\}$, and $R_2 = \{\mathtt{F}, \mathtt{G}, \mathtt{H}\}$. Again, no more vertices can be added without violating the balanced biclique constraint.

This problem has been proven to be $\mathcal{NP}$-hard in various works [2,9,11]. Some theoretical results proposed bounds for the maximum size that the optimal solution for the MBBP can have [6], and it has been proven to be hard to approximate within a certain factor [7].

Biclique graphs have been proven to be useful in several real-life applications, most of them in the field of biology: biclustering microarray data [4,17,18], optimization of the phylogenetic tree reconstruction [14], or identifying common gen-set associations [5], among others [3,12]. In particular, the MBBP has additional applications in a diverse set of fields: folding of programmable logic arrays in Very Large Scale Integration (VLSI) design [13], or new nanoelectronic systems design [1,15,16], among others.

Despite of the practical applications of the MBBP, not many efficient algorithms for it has been proposed, mainly due to the difficulty of the problem. However, if the MBBP does not consider that the solution must be balanced, the resulting problem is solvable in polynomial time [19], although most of the solutions obtained are totally unbalanced, making the results not adaptable for the problem under consideration.

Most of the previous approaches follow a destructive approach where the initial solution contains all the nodes and the heuristic iteratively removes nodes from L' and R' until the incumbent solution becomes feasible. The algorithms mainly differ in the order in which the vertices to be removed are selected. In particular, [15] selects the vertices with the largest degree, while [1] removes the vertex with the largest number of minimum degree nodes in the other layer. Additionally, some algorithms have tried to combine both criteria [20,21]. The best algorithm found in the literature consists of a evolutionary algorithm [22] that proposes a new mutation operator as well as a new local search method to improve the quality of the solutions generated.

The remaining of the paper is organized as follows: Sect. 2 describes the algorithmic proposal for the MBBP; Sect. 3 presents the experiments performed to evaluate the quality of the proposal; and Sect. 4 draws some conclusions on the research.

2 Reduced Variable Neighborhood Search

Variable Neighborhood Search (VNS) [10] is a metaheuristic framework based on systematic changes of neighborhoods. As a metaheuristic algorithm, it does not guarantee the optimality of the solutions obtained, but it is focused on obtaining high quality solutions in reasonable computing times. The constant evolution of VNS has resulted in several variants, among which we can highlight Basic VNS, Reduced VNS, Variable Neighborhood Descent, General VNS, Skewed VNS, Variable Neighborhood Decomposition Search, among others.

Most of the variants differ in the way of exploring the considered neighborhoods. Specifically, Variable Neighborhood Descent (VND) considers a totally deterministic exploration of the solution space, while the exploration performed by Reduced VNS (RVNS) is totally stochastic. Some other variants combine both deterministic and stochastic changes of neighborhoods (e.g., Basic VNS, General VNS).

As stated in previous works [21,22], designing a local search method for the MBBP is a very difficult task mainly due to the complexity of maintaining a feasible solution (i.e. a balanced biclique) after removing or adding vertices to a previous solution. In other words, the MBBP is not suitable for designing local search methods in order to find a local optimum with respect to a given solution. Therefore, this work proposes a Reduced VNS algorithm, which is based on a random exploration of the considered neighborhoods.

RVNS is a VNS variant useful for large instances in which local search is very time consuming, or for those problems in which it is not easy to design a local search method. Algorithm 1 presents the pseudocode of RVNS.

Algorithm 1. *RVNS*$(B, k_{\max}, t_{\max})$

```
1: repeat
2:     $k \leftarrow 1$
3:     while $k \leq k_{\max}$ do
4:         $B' \leftarrow \mathit{Shake}(B, k)$
5:         $k \leftarrow \mathit{NeighborhoodChange}(B, B', k)$
6:     end while
7: until $\mathit{CPUTime}() \leq t_{\max}$
8: return $B$
```

The algorithm requires three input parameters: B, the initial feasible solution, that can be randomly generated or using a more elaborated constructive procedure; $k_{\max}$, the maximum neighborhood to be explored; and $t_{\max}$, the maximum computing time in which RVNS is allowed to explore the search space. Each RVNS iteration starts from the initial neighborhood (step 2). Then, the method explores each one of the considered neighborhoods (steps 3–6) as follows. Firstly, the method generates a random solution B' in the current neighborhood k of the incumbent solution B using the *Shake* method (step 4). Then, the *NeighborhoodChange* method (step 5) is responsible for selecting the next neighborhood to be explored. Specifically, if the new solution B' is better than the incumbent one B, then it is updated ($B \leftarrow B'$), restarting the search from the initial neighborhood ($k \leftarrow 1$). Otherwise, the search continues with the next neighborhood ($k \leftarrow k + 1$). A RVNS iteration ends when reaching the maximum predefined neighborhood $k_{\max}$. It is worth mentioning that the maximum neighborhood considered in RVNS is usually small due to the random nature of the *Shake* method, since a large value for $k_{\max}$ would produce the same result as restarting the search from a new initial solution. The RVNS method is executed until reaching a maximum computing time $t_{\max}$.

2.1 Constructive Method

The initial solution for VNS can be generated at random or with a more elaborated constructive procedure. This work proposes a constructive procedure based on the Greedy Randomized Adaptive Search Procedure (GRASP) [8]. This methodology considers a greedy function that evaluates the importance of inserting a vertex in the solution under construction. Algorithm 2 presents the pseudocode of the constructive method proposed.

The method starts by randomly selecting a vertex from layer L (step 11), inserting it in the corresponding layer L' of the solution (step 12). Then, two candidate lists (CL) are created, one for each layer of the graph (steps 13–14).

Algorithm 2. *Construct*$(G = (L, R, E), \alpha)$

```
1: function UpdateLayer($CL_1, CL_2, \alpha$)
2:     $g_{\min} \leftarrow \min_{v \in CL_1} g(v)$
3:     $g_{\max} \leftarrow \max_{v \in CL_1} g(v)$
4:     $\mu \leftarrow g_{\min} + \alpha \cdot (g_{\max} - g_{\min})$
5:     $RCL \leftarrow \{v \in CL_1 \ ; \ g(v) \leq \mu\}$
6:     $v \leftarrow Random(RCL)$
7:     $CL_1 \leftarrow CL_1 \setminus \{v\}$
8:     $CL_2 \leftarrow CL_2 \setminus \{u \in CL_2 \ : \ (v, u) \notin E\}$
9:     return $v$
10: end function
11: $v \leftarrow Random(L)$
12: $L' \leftarrow \{v\}$
13: $CL_L \leftarrow L \setminus \{v\}$
14: $CL_R \leftarrow \{u \in R \ : \ (v, u) \in E\}$
15: while $CL_L \neq \emptyset$ and $CL_R \neq \emptyset$ do
16:     $v_r \leftarrow$ UpdateLayer($CL_R, CL_L, \alpha$)
17:     $R' \leftarrow R \cup \{v_r\}$
18:     $v_l \leftarrow$ UpdateLayer($CL_L, CL_R, \alpha$)
19:     $L' \leftarrow L \cup \{v_l\}$
20: end while
21: return $B = (L', R', \{(v, u) \ v \in L' \wedge u \in R'\})$
```

Notice that each candidate lists only contains those vertices from each layer that can be selected maintaining the solution feasible (i.e., the solution is a compete bipartite graph). On the one hand, in this first step, CL_L contains all the vertices from L excepting the selected vertex v. On the other hand, CL_R contains all adjacent vertices to v in R, since otherwise the solution would not be a bipartite complete graph. Then, the method iterates adding a vertex in layer R' and then in layer L', while there are still candidate vertices in both layers (steps 16–19).

The selection of the next vertex is described in function *UpdateLayer* that requires from three parameters: the candidate list from which the vertex must be selected, CL_1, the candidate list of the other layer, CL_2, and the α parameter that determines the greediness/randomness of the selection. A greedy function g that evaluates the quality of a candidate vertex v must be defined. In this work, we propose the number of adjacent vertices in the oppositive candidate list. More formally,

$$g(v, CL) = |\{u \in CL \ : \ (v, u) \in E\}|$$

The first step to select the next vertex consists of evaluating the minimum ($g_{\min}$) and maximum ($g_{\max}$) values for the greedy function value (steps 2–3). Then, a Restricted Candidate List (*RCL*) is constructed (step 5) with those vertices whose objective function value is larger or equal than a previously evaluated threshold μ (step 4). The values for the α parameter are in the range

0–1, where $\alpha = 0$ implies that the method is totally random, and $\alpha = 1$ transforms the algorithm in a completely greedy method. The next vertex is selected at random from the RCL (step 6), updating the corresponding candidate lists. In particular, CL_1 is updated by removing the selected vertex v from it, while CL_2 is updated by removing those vertices that are not adjacent to v, since they cannot be selected in further iterations maintaining the feasibility of the solution.

2.2 Shake

The *Shake* method is a stage inside VNS methodology designed to escape from local optima found during the search. It consists of randomly perturbing a solution with the aim of exploring a wider region of the solution space. This phase of the VNS methodology is focused on diversifying the search.

Given a neighborhood k, the *Shake* method proposed in this work removes k elements at random from each layer. The resulting solution is feasible but the value of the objective function is always smaller, since it reduces the number of vertices selected.

Considering the constraints of the problem, if a vertex v is included in the solution, then all the vertices of the opposite layer that are not adjacent to v cannot be included in future iterations, since the resulting solution would not be a biclique. However, the removal of some vertices in the *Shake* method can eventually allow new vertices to be included in the solution (i.e., those that were not adjacent to any of the vertices removed).

Therefore, we propose a reconstruction stage that is executed after each *Shake* method. In particular, the reconstruction phase tries to add new vertices to the solution, from those that were not feasible to add before executing the *Shake* method.

The reconstruction stage always improves the quality of the solution or, at least, maintains the quality of the solution produced by the *Shake* method. Notice that the reconstructed solution outperforms the initial one if and only if reconstruction stage is able to insert more than k vertices in each layer.

The random nature of this procedure makes it difficult to improve the quality of the solution. In order to mitigate this effect we consider four variants for the *Shake* method. These differ in how the destruction and reconstruction phases are performed. Each stage can be either random (R) or greedy (G), which leaves us with four variants shown in Table 1. For instance, the *Shake* variant GR firstly destroys the solution with a greedy selection of vertices and then reconstructs it randomly.

The reconstruction stage follows the same idea that the constructive method described in Sect. 2.1, with the parameters $\alpha = 0$ for the random construction (R) and $\alpha = 1$ for the greedy one (G). For the destruction stage we use a template similar to the constructive method, where we evaluate all candidates with a heuristic function, in this case those vertices already included in the solution, and then choose one of the more promising vertices. We need to define a new heuristic function g' that allows us to score the candidates for removal. Without

Table 1. Summary of the four variants considered for the *Shake* procedure.

Variant	Destruction	Reconstruction
RR	Random	Random
RG	Random	Greedy
GR	Greedy	Random
GG	Greedy	Greedy

loss of generality, for a given vertex $v \in S$ located in layer L, we calculate $g'(v)$ as the number of vertices in the opposite layer R not connected to v. More formally,

$$g'(v, S) = |\{u \in R \ : \ (u, v) \notin E\}|$$

3 Computational Results

In this section we present two sets of experiments: the preliminary experiments, performed to tune the parameters of the proposed algorithm; and the final experiment designed to test the quality of our proposal and compare it with the current state of the art. All the algorithms have been coded in Java 8 and were executed on a computer with an Intel i7 (7660U) CPU with 2.5 GHz and 8 GB RAM.

We use the same data set presented in [22], which consist of 30 instances with sizes $n = \{250, 500\}$ and different densities. This data set is used to compare the performance of our algorithm with the current state of the art.

In these experiments we report: the average size of the largest balanced biclique obtained, Avg. Size; the average execution time per instance, Avg. Time (s); the average percentage deviation to the best solution found in the experiment, %Dev.; and the number of times an algorithm reaches the best solution found for a given instance in the current experiment, # Best.

3.1 Preliminary Experiments

The following experiments are designed to select the best variant for the proposed algorithm. A small group of 6 representative instances, selected from the original 30, was used in these experiments to avoid overfitting the parameters to the data set in the final experiment.

The first experiment is designed for testing the effect of the α parameter in the constructive procedure, considering $\alpha = \{0.25, 0.50, 0.75, RND\}$. The *RND* value means that we will use a different α-value, selected randomly, in each iteration. The results in Table 2 show us that the best performance is achieved when alpha is selected randomly in each iteration. The procedure obtains on average a balanced biclique of 52.67 vertices and finds the best solutions among this experiment for all 6 instances. The results obtained shows that considering small values of α (i.e., increasing the randomness of the method) always results

Table 2. Comparison of the constructive method when considering different values for α.

α	Avg. size	Avg. time (s)	%Dev.	#Best
0.25	46.17	375.00	11.67	0
0.50	48.33	375.00	8.16	0
0.75	51.83	375.00	1.40	2
RND	**52.67**	**375.00**	**0.00**	**6**

in worse quality solutions. However, the *RND* value allows us to diversify the search by considering both small and large values of α, thus obtaining the best results in terms of average objective function value, average deviation, and total number of best solutions found. Therefore, in the following experiments we will use this configuration for the α parameter in our proposal.

Table 3. Comparison of the RVNS algorithm for a fixed neighborhood $K_{max} = 50$ and different variations of the shake procedure.

Shake	Avg. size	Avg. time (s)	%Dev.	#Best
RR	48.83	375.00	9.59	0
RG	**54.17**	375.00	0.00	6
GR	49.50	375.00	8.48	0
GG	53.17	375.00	2.07	0

The next experiment is designed to select the best variant for the shake procedure. We assume a $k_{max} = 0.5 \cdot n$ and consider the four variants of the shake procedure according the type of destruction, random (R) or greedy (G), and the type of reconstruction, random (R) or greedy (G), as presented in Sect. 2.2. We can see in Table 3 the results for this experiment and how the variant RG, random removal with greedy reconstruction, has the best performance. It achieves an average size of 54.14 and find better solutions than all other variants in all 6 instances. Notice that the best results are obtained when considering a greedy reconstruction, but the inclusion of the random destruction is able to reach better quality solutions than the greedy destruction.

In the last preliminary experiment we want to measure the impact of the maximum neighborhood explored in our algorithm. We use the best configuration of the previous experiments and test different neighborhoods $k_{max} = \{0.10, 0.20, 0.30, 0.40, 0.50\}$ for our *RVNS* framework. It is important to remark that a neighborhood k removes $k \cdot n$ vertices of the solution. In this experiment we can see that expanding the size of the neighborhood generally allows to reach better solutions as the %Dev. decreases. However, this improvement stagnates after $k_{\max} = 0.40$ where the algorithm reaches its maximum performance (Table 4).

Table 4. Comparison of the RVNS algorithm when considering different values for K_{max}.

K_{max}	Avg. cost	Avg. time (s)	%Dev.	#Best
0.10	53.50	375.00	1.45	2
0.20	53.50	375.00	1.17	2
0.30	53.83	375.00	0.62	4
0.40	**54.17**	375.00	0.00	6
0.50	54.17	375.00	0.00	6

Analyzing the preliminary experimentation, the best algorithm is configure with $\alpha = RND$ for the constructive procedure, the shake variant RG which considers random destruction and greedy reconstruction, and a maximum neighborhood of $k_{\max} = 0.40$.

3.2 Final Experiment

In the last experiment we compare our proposal with the best algorithm found in the state of the art [22] using the same set of 30 instances. In particular, it consists of a memetic algorithm that considers a local search based on structure mutation. RVNS is executed iteratively until reaching a time limit in seconds equal to three times the size of the current instance. Table 5 shows the results obtained when comparing the best variant of RVNS with the memetic algorithm (EA/SM). As it can be derived from the table, RVNS is able to find (on average) bicliques of just one node less than the bicliques obtained by the memetic algorithm. However, it has an execution time that is roughly half of the memetic algorithm.

Table 5. Comparison of the RVNS algorithm with the best in the state of art.

	Avg. size	Avg. time (s)	%Dev.	#Best
EA/SM	55.10	2075.11	0.04	29
RVNS	54.33	1125.00	1.71	10

4 Conclusions

This work analyzes the performance of Reduced VNS for generating high quality solutions for the Maximum Balanced Biclique Problem efficiently. Specifically, we propose an intensified shaking stage which is conformed by a destruction and reconstruction phase. The experiments performed show the relevance of performing these phases in a random or greedy manner. The results obtained show the possibilities of the RVNS proposal, obtaining, on average, solutions that are really close to the best ones found in the state of the art. Furthermore, the absence of a local search in the proposed algorithm allows it to require half of the computing time of the best algorithm found in the literature.

References

1. Al-Yamani, A.A., Ramsundar, S., Pradhan, D.K.: A defect tolerance scheme for nanotechnology circuits. IEEE Trans. Circuits Syst. **54–I**(11), 2402–2409 (2007)
2. Alon, N., Duke, R.A., Lefmann, H., Rödl, V., Yuster, R.: The algorithmic aspects of the regularity lemma. J. Algorithms **16**(1), 80–109 (1994)
3. Baker, E.J., et al.: Ontological discovery environment: a system for integrating gene-phenotype associations. Genomics **94**(6), 377–387 (2009)
4. Cheng, Y., Church, G.M.: Biclustering of expression data. In: Proceedings of the 8th ISMB, pp. 93–103. AAAI Press (2000)
5. Chesler, E.J., Langston, M.A.: Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data. In: Eskin, E., Ideker, T., Raphael, B., Workman, C. (eds.) RRG/RSB -2005. LNCS, vol. 4023, pp. 150–165. Springer, Heidelberg (2007)
6. Dawande, M., Keskinocak, P., Swaminathan, J.M., Tayur, S.: On bipartite and multipartite clique problems. J. Algorithms **41**(2), 388–403 (2001)
7. Feige, U., Kogan, S.: Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report (2004)
8. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. J. Global Optim. **6**(2), 109–133 (1995)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
10. Hansen, P., Mladenović, N.: Variable Neighborhood Search, pp. 313–337. Springer, Boston (2014)
11. Johnson, D.S.: The NP-completeness column: an ongoing guide. J. Algorithms **13**(3), 502–524 (1992)
12. Mushlin, R.A., Kershenbaum, A., Gallagher, S.T., Rebbeck, T.R.: A graph-theoretical approach for pattern discovery in epidemiological research. IBM Syst. J. **46**(1), 135–150 (2007)
13. Ravi, S.S., Lloyd, E.L.: The complexity of near-optimal programmable logic array folding. SIAM J. Comput. **17**(4), 696–710 (1988)
14. Sanderson, M.J., Driskell, A.C., Ree, R.H., Eulenstein, O., Langley, S.: Obtaining maximal concatenated phylogenetic data sets from large sequence databases. Mol. Biol. Evol. **20**(7), 1036–1042 (2003)
15. Tahoori, M.B.: Application-independent defect tolerance of reconfigurable nanoarchitectures. JETC **2**(3), 197–218 (2006)
16. Tahoori, M.B.: Low-overhead defect tolerance in crossbar nanoarchitectures. JETC **5**(2), 11 (2009)
17. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. In: ISMB, pp. 136–144 (2002)
18. Wang, H., Wang, W., Yang, J., Yu, P.S.: Clustering by pattern similarity in large data sets. In: Franklin, M.J., Moon, B., Ailamaki, A. (eds.) SIGMOD Conference, pp. 394–405. ACM (2002)

19. Yannakakis, M.: Node-deletion problems on bipartite graphs. SIAM J. Comput. **10**(2), 310–327 (1981)
20. Yuan, B., Li, B.: A low time complexity defect-tolerance algorithm for nanoelectronic crossbar. In: International Conference on Information Science and Technology, pp. 143–148 (2011)
21. Yuan, B., Li, B.: A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. ACM J. Emerg. Technol. Comput. Syst. (JETC) **10**(3), 25 (2014)
22. Yuan, B., Li, B., Chen, H., Yao, X.: A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem. IEEE Trans. Cybern. **45**(5), 1040–1053 (2015)