Information Processing Letters 112 (2012) 535–539

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

An exact exponential time algorithm for counting bipartite cliques ☆

Konstantin Kutzkov

IT University of Copenhagen, Denmark

ARTICLE INFO

Article history:
Received 19 November 2011
Received in revised form 4 April 2012
Accepted 4 April 2012
Available online 10 April 2012
Communicated by Ł. Kowalik

Keywords:
Analysis of algorithms
Exact exponential time algorithms
Counting bipartite cliques

ABSTRACT

We present a simple exact algorithm for counting bicliques of given size in a bipartite graph on n vertices. We achieve running time of $O(1.2491^n)$, improving upon known exact algorithms for finding and counting bipartite cliques.

1. Introduction

Many relationships between real world objects can be abstractly modeled as graphs. Often we are interested in dense subgraphs capturing important information. One of the most studied examples of such dense subgraphs is the clique problem asking for a subset of the graph's vertices such that any two of them are connected by an edge.

A recent trend in algorithmic research has been to design fast exponential time algorithms solving hard problems exactly when approximate solutions are either not satisfying or impossible. New algorithms have been designed considerably increasing the size of efficiently computable instances of hard problems. Notable examples include solving k-colorability for graphs on n vertices in time and space $O^*(2^n)$ independent of k [4] and k-SATISFIABILITY on n variables in time $O^*((\frac{2(k-1)}{k})^n)$ and polynomial space [13,14].[1]

The problem of finding a clique of size k in a graph G is equivalent to the canonical NP-complete problem of finding an independent set of size k in the complement of G, i.e. the graph obtained from G with an edge between two vertices u and v if and only if there is no edge between u and v in G. The best known polynomial space exact algorithm for the maximum independent set problem runs in time $O^*(1.2114^n)$ [5] and for counting maximum independent sets in time $O(1.2377^n)$ [16].

Algorithms and hardness results have shed light on the computational complexity of the problem of finding and counting bipartite cliques when the problem is restricted to bipartite graphs [9,12,15]. We give a brief overview on published exact exponential time algorithms for the two problems.

Given a bipartite graph $G = (L \cup R, E)$ Fernau and Niedermeier [7] present a sophisticated branching algorithm for finding a vertex cover with at most t_l vertices in L and t_r vertices in R running in time $O^*(1.3999^{(t_l+t_r)})$. Note that this is equivalent to finding a bipartite independent set with at least $|L| - t_l$ vertices in L and $|R| - t_r$ vertices in R. Binkele-Raible et al. [3] use this observation to design an exact algorithm for finding a (k_1, k_2)-biclique running in time $O^*(1.30052^n)$. Their algorithm is simple and intuitive but uses as a subroutine the algorithm from [7]. Recently, Couturier and Kratsch [6] presented an exact algorithm for finding bipartite cliques running in time $O(1.2691^n)$ and exponential space. As for counting bicliques, we are aware of only one non-trivial algorithm. Gaspers et al. [10] present an exact algorithm for counting *maximal* bicliques in general graphs in time

☆ This research was partially supported by the Swedish Research Council grant VR 2007-6595, Exact Algorithms.

E-mail address: konk@itu.dk.

[1] The O^* notation ignores polynomial factors.

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2012.04.001

$O(1.3642^n)$ and polynomial space. The algorithm can be easily extended to counting bicliques of given size in bipartite graphs. Note that the requirement on the bicliques to be maximal makes the problem inherently more difficult, see Section 4.1 of [10] for a discussion. The problem remains #P-hard even when restricted to planar bipartite graphs of bounded degree [15].

Bicliques naturally arise in many areas like artificial intelligence, computational biology, data mining, etc. Consider the fundamental knowledge discovery problem of frequent pattern mining [1]. We are given a set m of transactions each containing a subset of items $i \in \mathcal{I}$ for some ground set $\mathcal{I}$. For example transactions can represent market baskets and we are interested in finding patterns among purchased items such as "customers who buy beer are likely to also buy fish fingers". We want such patterns to be representative, thus one sets a support threshold indicating how many times the items in question appear together in a transaction. It turns out that the computationally expensive step is to mine such *α-frequent k-itemsets*, i.e. sets of k items appearing together in at least αm transactions for $k > 1$ and $\alpha > 0$ [11]. The problem can be naturally reduced to listing bicliques in a bipartite graph by associating items and transactions with left and right-hand side vertices, respectively, and then enumerating all bicliques with k left-side vertices and at least αm right-side vertices. Often a low support threshold causes a combinatorial explosion in the number of frequent k-itemsets, thus a faster counting algorithm is necessary for fine-tuning the input parameters of a frequent pattern mining algorithm. We refer the reader to [2] for a list of other applications.

2. Preliminaries

Notation. Let $G = (V, E)$ be a simple undirected graph on n vertices and m edges. For an edge $(u, v) \in E$ v is a *neighbor* of u, and the set of neighbors of u is $N(u)$ and $N[u] = N(u) \cup \{u\}$. The *degree* of a vertex u is the number of its neighbors and the degree of G is the maximum vertex degree in G. A *tree* is a connected graph without cycles, a *forest* is a collection of pairwise disjoint trees. We define a *parent–child* relationship on a tree such that each vertex has at most one parent. A vertex without a parent is called a *root* and a vertex with no children is called a *leaf*. A tree has exactly one root. A vertex u is an *ancestor* of a vertex v and the vertex v is a *descendant* of u if u is on the path from v to the root. The *depth* of a vertex v is the length of the path from the root to v and the *level* of v is the length of the longest path from v to a leaf which is a descendant of v.

A *vertex cover* of G is a subset of vertices $C \subseteq V$ such that for each $(u, v) \in E$, $u \in C$ or $v \in C$ holds. The complement set $V \backslash C$ is an *independent set* in G. The graph obtained by removing a vertex v from G and all its adjacent edges is denoted as $G \backslash \{v\}$.

In a bipartite graph $B = (L \cup R, E)$ for all edges $(x, y) \in E$, $x \in L$ iff $y \in R$. We will refer to vertices in L and R as *left-side*, respectively *right-side*, vertices. A (k_l, k_r)-*bipartite clique* (or *biclique*) in $B = (L \cup R, E)$ is a subgraph $K = (L_K \cup R_K, E_K)$ of B, such that $L_K \subseteq L$, $R_K \subseteq R$, $E_K \subseteq E$ and for all $v_L \in L_K$, $v_R \in R_K$ it holds $(v_L, v_R) \in E$ and $|L_K| \geqslant k_l$, $|R_K| \geqslant k_r$. Similarly, a (t_l, t_r)-*bipartite vertex cover* covers B with at most t_l vertices from L and at most t_r vertices from R and in a (t_l, t_r)-*bipartite independent set* there are at least t_l left vertices not connected to any of t_r right vertices. Since a tree does not contain a cycle a tree is also a bipartite graph. A *bipartite complement* $\bar{B} = (L \cup R, \bar{E})$ of a bipartite graph $B = (L \cup R, E)$ is a bipartite graph such that $(u, v) \in \bar{E}$ iff $(u, v) \notin E$ for $u \in L$, $v \in R$. It is easy to see that a (k_l, k_r)-biclique in B is a (k_l, k_r)-bipartite independent set in $\bar{B}$. If the set $I \subseteq V$ is a (k_l, k_r)-independent set in B then $V \backslash I$ is a (t_l, t_r)-bipartite vertex cover in B with $t_l := |L| - k_l$ and $t_r := |R| - k_r$. The problem of counting the number of bipartite (t_l, t_r)-vertex covers in a graph is denoted as #BVC(G, t_l, t_r).

The algorithm is based on *branching* on a vertex $u \in G$ distinguishing the cases when u is either taken in a bipartite vertex cover or not. The first case is denoted by $G[u]$ and the second by $G \backslash \{u\}$.

Signatures. For a given bipartite graph G a bipartite vertex cover with exactly t_l left vertices and t_r right vertices has a *signature* $\sigma_G(t_l, t_r)$. The *weighted signature* of G, $\sigma_G(t_l, t_r, c_G)$, denotes the number c_G of distinct bipartite vertex covers of G with exactly t_l vertices in L and t_r vertices in R. We assume that an empty graph, denoted as *nil*, has only one signature $\sigma_{nil}(0, 0)$. We say that two signatures for a given graph are *identical* if they correspond to covers with equal number of left and right vertices. The *product* of two weighted signatures $\sigma_{G_1}(t_l^{(G_1)}, t_r^{(G_1)}, c_{G_1})$ and $\sigma_{G_2}(t_l^{(G_2)}, t_r^{(G_2)}, c_{G_2})$ is the new signature $\sigma_G(t_l^{(G_1)} + t_l^{(G_2)}, t_r^{(G_1)} + t_r^{(G_2)}, c_{G_1} \cdot c_{G_2})$ where $G = (V_1 \cup V_2, E_1 \cup E_2)$ is the union of the connected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. For a given #BVC(G, t_l, t_r) problem instance we will say that the signature $\sigma_{G'}(q_l, q_r)$ is *admissible* if $q_l \leqslant t_l$ and $q_r \leqslant t_r$ for a graph $G' \subseteq G$.

Colored graphs. In our algorithm we partition the set of edges E in two subsets of black and white edges, B and W, such that $E = B \cup W$ and $B \cap W = \emptyset$. We maintain the invariant that edges in W form a forest. We call the forest formed by W *proper* if in each tree only the root is possibly adjacent to any black edges and assume that an empty forest is proper. The subgraphs induced by the edge sets B and W are the subgraphs of G containing only edges from B and W, denoted as $G_B = (V, B)$ and $G_W = (V, W)$, respectively. By coloring a black edge e white we mean the operation $B = B \backslash \{e\}$ and $W = W \cup \{e\}$ for $e \in B$.

3. The #BVC(G, t_l, t_r) problem on trees

Lemma 1. *Let $F = (V, E)$ be a forest. Then #BVC(F, t_l, t_r) can be solved in polynomial time in the size of the forest.*

Proof. We present an algorithm for counting the number of (t_l, t_r)-bipartite vertex covers using a standard dynamic programming approach. Assume F consists of a single tree T and the depth of T is d. Starting with the nodes of depth d for each node v of the tree we store a set of signatures for bipartite vertex covers for the subtree rooted

at v with no more than t_l left vertices and t_r right vertices. We divide the signatures in two sets S_v and $S_{\bar{v}}$ depending on whether the vertex v is part of the cover or not. Once the signatures of all nodes of depth d have been computed, we go one level higher in the tree and compute the signatures of the nodes at depth $d-1$.

We show now how to compute the signatures for the bipartite vertex covers of a tree rooted at v with already computed signatures for all its children. Assume w.l.o.g. that v is a left-side vertex and we want to find all signatures for covers containing v. We start with an empty forest F and an empty set of signatures S_F. We first add the weighted signature $\sigma_{nil}(0,0,1)$ to S_F. Let v have k children. Assuming an order on v's children, we consider the ith subtree T_i, $1 \leqslant i \leqslant k$, compute the product of each weighted signature in S_F, with each weighted signature of covers of T_i and update S_F to contain only the new signatures. Now we look through S_F and compress all identical weighted signatures by adding their corresponding counters and remove all weighted signatures of covers which cannot be extended to a (t_l, t_r)-bipartite vertex cover. At the end we add 1 to the number of left vertices in each signature in S_F and set $S_v := S_F$. For computing the set of weighted signatures not including v, $S_{\bar{v}}$, we observe that only signatures including v's children can correspond to a vertex cover for the subtree rooted at v. Therefore we have to repeat the above procedure but only considering the signatures S_u for trees rooted at u, u being a child of v. If S_v and $S_{\bar{v}}$ are empty the algorithm returns 0. At the end we return the sum of the counters of the weighted signatures in S_r and $S_{\bar{r}}$ for the root of the tree r.

We show that S_r and $S_{\bar{r}}$ will contain exactly one signature for each (q_l, q_r)-bipartite vertex cover such that $q_l \leqslant t_l$ and $q_r \leqslant t_r$ by induction on the structure of the tree. Consider a vertex v. If v a leaf then correctness is trivial. Assume we have computed the weighted signatures for v's children and let denote by F the forest obtained by the trees rooted at v's children. In particular, for a given weighted signature $\sigma_{T_u}(t_l^{(T_u)}, t_r^{(T_u)}, c_{T_u})$ the counter c_{T_u} records the number of unique bipartite vertex covers of given size for the tree $T_u \in F$. Let $\sigma_F(t_l^{(F)}, t_r^{(F)}, c_F) \in S_F$ be an admissible weighted signature computed after all trees in F have been considered. We compute the products of signatures for covers of pairwise disjoint trees, therefore the counter c_F will record the number of unique covers for F. Further, assuming that each (q_l, q_r)-bipartite vertex, $q_l \leqslant t_l$, $q_r \leqslant t_r$, is recorded in a signature we see that each such cover for F will contribute to the counter of exactly one signature. We consider the cases when v is either included in or excluded from the cover, therefore each signature added to either S_v or $S_{\bar{v}}$ corresponds to a unique (q_l, q_r)-bipartite vertex cover, $q_l \leqslant t_l$ and $q_r \leqslant t_r$, and each such cover is recorded in an admissible signature.

For the running time analysis observe that the number of signatures for (t_l, t_r)-bipartite covers is bounded by $(t_l+1)\cdot(t_r+1)$. For a parent vertex v with k children we compute the signatures for T_v from the signatures of its children in $O(k \cdot t_l^2 \cdot t_r^2)$. Since k, t_l, t_r are upper bounded by the number of vertices in the tree and we compute exactly once the signatures for a subtree rooted at a given vertex the claim follows.

The algorithm trivially extends to counting bipartite cliques in forests. □

Lemma 2. *Let $G = (L \cup R, E)$, $E = (B \cup W)$, be a colored bipartite graph and G_W a proper forest. Let v be a vertex with only one adjacent black edge $e = (u, v)$. Then after coloring e white G_W remains a proper forest.*

Proof. We show that $W \cup \{e\}$ is a proper forest. Since v is adjacent to a black edge it must hold that v is the root of a (possibly empty) tree in G_W, the forest induced by W, call it T_v. By coloring e white v is not any more adjacent to a black edge. If u were not adjacent to another tree in G_W then clearly $e \cup T_v$ is a tree rooted at u. Otherwise since u was adjacent to a black edge, namely e, it must hold that u is the root of one or more other trees of white edges. Thus, by coloring e white T_v becomes a subtree at u. Since the trees in G_W are pairwise disjoint and e causes the creation of only one new tree, it follows that all trees in $W \cup \{e\}$ are pairwise disjoint. Also, in the new tree only u is possibly adjacent to a black edge, thus $W \cup \{e\}$ induces a proper forest. □

Corollary 1. *For a colored bipartite graph $G = (L \cup R, B \cup W)$ such that no vertex is adjacent to more than two black edges we can obtain a list of admissible signatures for #BVC(G, t_l, t_r) in polynomial time.*

Proof. G_B is a collection of paths and cycles. By branching on a vertex in each "black cycle" we obtain two trees for which we compute a set of admissible signatures in polynomial time and the set of all admissible signatures for #BVC(G, t_l, t_r) can then be obtained by the approach outlined in Lemma 1. □

4. Analysis of the algorithm

A pseudocode description of the algorithm CountBVC is presented in Fig. 1.

Correctness of CountBVC. First, we compress the already computed signatures as outlined in the proof of Lemma 1. If G consists of several connected components, we compute the signatures for each component and then the list of signatures for G as in the proof of Lemma 1. For each branching on a vertex v we partition the set of (t_l, t_r)-bipartite vertex covers in two disjoint subsets either including or excluding v. Further, when excluding a vertex all its neighbors must be in the cover. As shown in Lemma 2 by coloring white a black edge (u, v), such that v is not adjacent to any other black edge, we maintain the invariant that G_W is a proper forest. Therefore, if there are no vertices adjacent to more than two black edges we obtain a list of admissible signatures by dynamic programming.

Running time of CountBVC. We analyze the running time of our algorithm with the Measure & Conquer technique [8]. We assign a weight $0 \leqslant w_d \leqslant 1$ to each vertex of degree d in G. For a colored bipartite graph

function CountBVC
Input: colored bipartite graph $G = (L \cup R, B \cup W)$, positive integers t_l, t_r, a list of signatures $\mathcal{L}$.
Output: A list of admissible signatures for #BVC(G, t_l, t_r).
1: Compress all signatures in $\mathcal{L}$ as discussed in the proof of Lemma 1.
2: **if** $t_l < 0$ or $t_r < 0$ **then**
3: return $\mathcal{L}$
4: **if** there are more than one connected components in G **then**
5: Call CountBVC(G_i, t_l, t_r) for each component G_i and compute a list of admissible signatures for G from the signature lists $\mathcal{L}_i$ for each G_i.
6: **while** there exists a vertex adjacent to only one edge e in B **do**
7: $B = B \backslash \{e\}$
8: $W = W \cup \{e\}$
9: **if** all vertices are adjacent to at most two edges in B **then**
10: solve #BVC(t_l, t_r) by the dynamic programming algorithm from Corollary 1.
11: **else**
12: choose a vertex u with a maximum number of adjacent black edges, giving preference to vertices with neighbors of lower degree, let w.l.o.g. $u \in L$. Return CountBVC$(G \backslash \{u\}, t_l - 1, t_r,$ add-left$(\mathcal{L}, 1)) \cup$ CountBVC$(G \backslash N[u], t_l, t_r - |N(u)|,$ add-right$(\mathcal{L}, |N(u)|))$.

Fig. 1. A high-level pseudocode description of the algorithm. The algorithm recursively builds a search tree by branching on a vertex of degree at least two in G_B and returns a list of admissible signatures for #BVC(G, t_l, t_r). For a bipartite graph $G = (L \cup R, E)$ the algorithm is initially called as CountBVC$(G, L \cup R, E \cup \emptyset, t_l, t_r, nil)$, *nil* denotes the empty list. The functions add-left$(\mathcal{L}, i)$ and add-right$(\mathcal{L}, i)$ add i to the number of left and right vertices in each signature in the list $\mathcal{L}$, respectively, and return $\mathcal{L}$.

$G = (L \cup R, B \cup W)$ the complexity measure is $\mu(G) = \sum_{d \geqslant 0} n_d w_d$ where n_d is the number of vertices adjacent to exactly d black edges. In the following we will refer to the degree of a vertex v in G_B simply as the degree of v. Clearly, $\mu(G) \leqslant n$ therefore an upper bound on the running time with respect to $\mu(G)$ implies an upper bound with respect to the number of vertices n. As one can see in case $W = \emptyset$ the measure setting $w_d = 1$ for all d is the standard measure "number of vertices". We can set $w_0 = w_1 = 0$ because vertices adjacent to at most one black edge cannot exist in G_B after termination of the while-loop in lines 3–5 and its execution takes only polynomial time. We write $\Delta_i := w_i - w_{i-1}$. The Δ_i measure the progress made by the algorithm after assigning a vertex and decreasing the weight of its neighbors. In order to simplify the analysis we will enforce also the following constraint on the weights: $\Delta_2/2 \geqslant \Delta_3 \geqslant \cdots \geqslant \Delta_{i-1} \geqslant \Delta_i = 0$ for some $i > 2$ that has to be determined later. The reason for first condition $\Delta_2/2 \geqslant \Delta_3$ will become clear from the proof of the following

Lemma 3. *For each branching on a vertex u of maximum degree d in G_B the corresponding recurrence is majorized by $T(\mu(G)) \leqslant T(\mu(G) - w_d - \sum_{v \in N(u)} \Delta_{i^v}) + T(\mu(G) - w_d - \sum_{v \in N(u)} w_{i^v} - \sum_{v \in N(u)} (i^v - 1)\Delta_d)$ where the degree of u's neighbor v is $2 \leqslant i^v \leqslant d$.*

Proof. We need to lower bound the total decrease of the weights of the neighbors of u's neighbors in G_B. After including u's neighbor v of degree i in the cover we delete $(i-1)$ black edges different from (u, v). The only possibility the deletion of a black edge does not reduce the weight of a vertex is when all of its adjacent edges are deleted and the deletion of one of them does not count since it corresponds to decreasing the weight of the vertex from w_1 to w_0. All vertices in G_B have degree at least 2, thus from the constraint $\Delta_2/2 \geqslant \Delta_3 \geqslant \cdots \geqslant \Delta_{i-1} \geqslant \Delta_i = 0$ and $d(v) \geqslant 2$ $\forall v \in B$ we observe that after deleting k distinct edges the decrease is at least $k\Delta_d$. □

Given the lists of admissible signatures for several connected components G_i, we can obtain in polynomial time the list of admissible signatures for the union of all G_i. Therefore, in the following we assume the graph consists of a single connected component. When removing a vertex from G_B we always decrease the degree of some other vertex in G_B. Thus, for a given connected component we can exclude from consideration the case where we have a d-regular bipartite graph G_B. This case occurs at most once in a given path in the search tree, thus for fixed d such cases only contribute a constant factor to the total running time implying that in line 12 we will (almost) always branch on a vertex of maximum degree with neighbors of lower degree in G_B.

Theorem 1. *For a bipartite graph $G = (L \cup R, E)$ on n vertices the number of (k_l, k_r)-bicliques can be computed in $O(1.2491^n)$ time and polynomial space.*

Proof. An algorithm counting the number of $(|L| - k_l, |R| - k_r)$-bipartite vertex covers also computes the number of (k_l, k_r)-bicliques. After CountBVC terminates, we can obtain the number of covers of admissible size in polynomial time from the list of its signatures. We give an upper bound on the number of recursive calls of CountBVC. Assume first all vertices in G_B are of degree at most 5. By enumerating the possible combinations we computed an approximation of the weights w_i yielding the claimed upper bound on the running time: $w_0 = w_1 = 0$, $w_2 = 0.620707$, $w_3 = 0.881125$, $w_4 = 0.967413$, $w_5 = 0.997178$, $w_d = 1.0$ for $d \geqslant 6$. It can now be easily verified by plugging the recurrence from Lemma 3 into the possible cases when G_B has degree at most 5 that $O(1.2491^{\mu(G)})$ is an upper bound on the running time of the algorithm. As shown in [8] each recurrence from a branching on a vertex of degree more than 5 in G_B majorizes a certain branching on a vertex of degree at most 5 in G_B, thus the bound follows. □

As observed by Binkele-Raible et al. [3] the problem of finding a non-induced biclique of a given size in a graph $G = (V, E)$ can be reduced to that of finding a biclique in a bipartite graph $G' = (V \cup V', E')$ such V' is a copy of V and $(u, v') \in E'$ iff $(u, v) \in E$. Thus, there is a one-to-

one correspondence between a biclique in G and a biclique in G'. Since G' has $2n$ vertices we obtain the following

Corollary 2. *The number of non-induced bipartite cliques in a graph $G = (V, E)$ on n vertices can be computed in time $O(1.561^n)$.*

Acknowledgements

The author would like to thank two anonymous referees for many helpful comments and suggestions and Nina Taslaman for discussions on graph theory terminology.

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: VLDB, Morgan Kaufmann, 1994, pp. 487–499.
[2] J. Amilhastre, M.-C. Vilarem, P. Janssen, Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs, Discrete Appl. Math. 86 (1998) 125–144.
[3] D. Binkele-Raible, H. Fernau, S. Gaspers, M. Liedloff, Exact exponential-time algorithms for finding bicliques, Inform. Process. Lett. 111 (2) (2010) 64–67.
[4] A. Björklund, T. Husfeldt, M. Koivisto, Set partitioning via inclusion–exclusion, SIAM J. Comput. 39 (2) (2009) 546–563.
[5] N. Bourgeois, B. Escoffier, V.Th. Paschos, J.M.M. van Rooij, Fast algorithms for max independent set, Algorithmica 62 (1–2) (2012) 382–415.
[6] J.-F. Couturier, D. Kratsch, Bicolored independent sets and bicliques, in: CTW 2011, pp. 130–133.
[7] H. Fernau, R. Niedermeier, An efficient exact algorithm for constraint bipartite vertex cover, J. Algorithms 38 (2) (2001) 374–410.
[8] F.V. Fomin, F. Grandoni, D. Kratsch, A measure & conquer approach for the analysis of exact algorithms, J. ACM 56 (5) (2009).
[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.
[10] S. Gaspers, D. Kratsch, M. Liedloff, On independent sets and bicliques in graphs, Algorithmica 62 (3–4) (2012) 637–658.
[11] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, 2000.
[12] D.S. Hochbaum, Approximating clique and biclique problems, J. Algorithms 29 (1998) 174–200.
[13] R.A. Moser, D. Scheder, A full derandomization of Schöning's k-SAT algorithm, in: STOC 2011, pp. 245–252.
[14] U. Schöning, A probabilistic algorithm for k-SAT based on limited local search and restart, Algorithmica 32 (4) (2002) 615–623.
[15] S.P. Vadhan, The complexity of counting in sparse, regular, and planar graphs, SIAM J. Comput. 31 (2) (2001) 398–427.
[16] M. Wahlström, A tighter bound for counting max-weight solutions to 2SAT instances, in: IWPEC 2008, pp. 202–213.