# An Exact Algorithm Based on Chain Implication for the Min-CVCB Problem*

Jianxin Wang, Xiaoshuang Xu, and Yunlong Liu

College of Information Science and Engineering, Central South University
Changsha 410083, China
jxwang@mail.csu.edu.cn, xxshuang@126.com, ylonglew@yahoo.com.cn

**Abstract.** The constrained minimum vertex cover problem on bipartite graphs (the Min-CVCB problem), with important applications in the study of reconfigurable arrays in VLSI design, is an NP-hard problem and has attracted considerable attention in the literature. Based on a deeper and more careful analysis on the structures of bipartite graphs, we develop an exact algorithm of running time $O((k_u+k_l)|G|+1.1892^{k_u+k_l})$, which improves the best previous algorithm of running time $O((k_u+k_l)|G|+1.26^{k_u+k_l})$ for the problem.

## 1 Introduction

With the development of VLSI technology, the scale of electric circuit chip becomes larger and larger, and the possibility of introducing defects also increases along with the manufacture craft. With the increasing in the chip integration, it is not allowed that the wrong memory element appears in the manufacture process. A better solution is to use reconfigurable arrays. A typical reconfigurable memory array consists of a rectangular array plus a set of $k_u$ spare rows and $k_l$ spare columns. A defective element is repaired by replacing the row or the column containing the element with a spare row or a spare column. Therefore, to repair a reconfigurable array with defective elements, we need to decide how the rows and columns in the array are selected and replaced by spare rows and columns. The constraint here is that we only have $k_u$ spare rows and $k_l$ spare columns. It has now become well-known that this problem can be formulated as a constrained minimum vertex cover problem on bipartite graphs [1], as follows.

**Definition 1 (Constrained minimum vertex cover in bipartite graphs (Min-CVCB)).** *Given a bipartite graph $G=(V,E)$ with the vertex bipartition $V=U\cup L$ and two integers $k_u$ and $k_l$, determine whether there is a minimum vertex cover of $G$ with at most $k_u$ vertices in $U$ and at most $k_l$ vertices in $L$.*

* This research is supported by the National Natural Science Foundation of China (60433020), the Program for New Century Excellent Talents in University (NCET-05-0683) and the Program for Changjiang Scholars and Innovative Research Team in University (IRT0661)

A. Dress, Y. Xu, and B. Zhu (Eds.): COCOA 2007, LNCS 4616, pp. 343–353, 2007.

The problem is NP-complete [9], therefore has no efficient algorithms in general. On the other hand, in practice the number of spare rows and spare columns is much smaller than the size of the reconfigurable array: typically, a reconfigurable array is a $1000 \times 1000$ matrix plus 20 spare rows and 20 spare columns [1]. Therefore, it is practically important, and theoretically interesting, to develop efficient algorithms for the Min-CVCB problem, assuming $k_u$ and $k_l$ much smaller than the size of the graph $G$.

Hasan and Liu [1] introduced the concept of critical set to develop a branch-and-bound algorithm for solving the Min-CVCB problem, based on the $A^*$ algorithm [2]. No explicit analysis was given in [1] for the running time of the algorithm, but it is not hard to see that in the worst-case the running time of the algorithm is at least of order of $2^{k_u+k_l} + mn^{1/2}$. Following the work in [1], the Min-CVCB problem has been extensively studied in last two decades. Most of these studies were focused on heuristic algorithms for the problem [3-6].

More recently, people have become interested in developing parameterized algorithms for the Min-CVCB problem [8-9]. Fernau and Niedermeier [8] used a branching search technology and developed an algorithm with running time $O((k_u + k_l)n + 1.3999^{k_u+k_l})$ for the problem. Chen and Kanj [9] proved that the Min-CVCB problem is NP-complete, and developed an improved algorithm of running time $O((k_u + k_l)|G| + 1.26^{k_u+k_l})$ for the problem. The algorithm given in [9] made use of a number of classical results in matching theory and recently developed techniques in parameterized algorithms, which is currently the best algorithm for the problem.

In this paper, we perform a deeper and more careful analysis on related structures of bipartite graphs. Based on the analysis, we effectively integrate the techniques of chain implication, branching search, and dynamic programming, and develop an improved parameterized algorithm EACI of running time $O((k_u + k_l)|G| + 1.1892^{k_u+k_l})$ for the Min-CVCB problem.

## 2 Related Lemmas

For further discussion of our algorithm EACI, we first give some definitions and describe certain known results that are related to the Min-CVCB problem and to our algorithm.

**Definition 2 (Bipartite graph).** *A graph $G$ is bipartite if its vertex set can be partitioned into two sets $U$ (the "upper part") and $L$ (the "lower part") such that every edge in $G$ has one endpoint in $U$ and the other endpoint in $L$. A bipartite graph is written as $G = (U \cup L, E)$ to indicate the vertex bipartition. The vertex sets $U$ and $L$ are called the $U$-part and the $L$-part of the graph. A vertex is a $U$-vertex (resp. an $L$-vertex) if it is in the $U$-part (resp. the $L$-part) of the graph.*

Let $G = (U \cup L, E)$ be a bipartite graph with a perfect matching. The graph $G$ is *elementary* if every edge in $G$ is contained in a perfect matching in $G$. It is known that an elementary bipartite graph has exactly two minimum vertex covers, namely $U$ and $L$, without any other possibility [10].

**Lemma 1.** [9] *The time complexity for solving an instance* $< G; k_u, k_l >$ *of Min-CVCB problem, where* $G$ *is a bipartite graph of* $n$ *vertices and* $m$ *edges, is bounded by* $O(mn^{1/2} + t(k_u + k_l))$*, where* $t(k_u + k_l)$ *is the time complexity for solving an instance* $< G'; k'_u, k'_l >$ *of Min-CVCB, with* $k'_u < k_u$*,* $k'_l < k_l$ *and* $G'$ *having perfect matchings and containing at most* $2(k'_u + k'_l)$ *vertices.*

**Lemma 2.** (The Dulmage-Mendelsohn Decomposition theorem [10]). *A bipartite graph* $G = (U \cup L, E)$ *with perfect matchings can be decomposed and indexed into elementary subgraphs* $B_i = (U_i \cup L_i, E_i)$*,* $i = 1, 2, \ldots r$*, such that every edge in* $G$ *from a subgraph* $B_i$ *to a subgraph* $B_j$ *with* $i < j$ *must have one endpoint in the* $U$*-part of* $B_i$ *and the other endpoint in the* $L$*-part of* $B_j$*. Such a decomposition can be constructed in time* $O(|E|^2)$*.*

The elementary subgraphs $B_i$ will be called (elementary) *blocks*. The block $B_i$ is a *d-block* if $|U_i| = |L_i| = d$. Edges connecting vertices in two different blocks will be called *inter-block edges*. Let $B_i$ be a block. The number $\lambda_{in}$ of blocks $B_j$ such that $i \neq j$ and there is an inter-block edge from the $U$-part of $B_i$ to the $L$-part of $B_j$ is called the *in-degree* of $B_i$. Similarly, the number $\lambda_{out}$ of blocks $B_j$ such that $i \neq j$ and there is an inter-block edge from the $U$-part of $B_j$ to the $L$-part of $B_i$ is called the *out-degree* of $B_i$.

**Lemma 3.** [10] *Let* $G$ *be a bipartite graph with perfect matchings, and let* $B_1$*,* $\ldots$*,* $B_r$ *be the blocks of* $G$ *given by the Dulmage-Mendelsohn Decomposition. Then any minimum vertex cover for* $G$ *is the union of minimum vertex covers of the blocks* $B_1, B_2, \ldots, B_r$*.*

By Lemma 1, in order to solve a general instance $\langle G; k_u, k_l \rangle$ of the Min-CVCB problem, we only need to concentrate on a "normalized" instance $\langle G'; k'_u, k'_l \rangle$ of the problem, in which $G'$ has a perfect matching and contains at most $2(k'_u + k'_l)$ vertices. By Lemma 2, the graph $G'$ with perfect matchings can be decomposed and represented as a directed acyclic graph (DAG) $D$ in which each node corresponds to a block in $G'$ and each edge corresponds to a group of inter-block edges from the $U$-part of a block to the $L$-part of another block. By Lemma 3, a minimum vertex cover of the graph $G'$ is the union of minimum vertex covers of the blocks $B_1$, $\ldots$, $B_r$. All these are very helpful and useful when we construct a desired minimum vertex cover in the originally given bipartite graph $G$.

## 3 The Strategy for Reducing the Search Space in Algorithm EACI

Algorithm EACI is based on the DAG D constructed above and its execution is depicted by a search tree whose leaves correspond to the potential constrained minimum vertex covers $K$ (shortly $K$) of the graph $G$ with at most $k_u$ $U$-vertices and at most $k_l$ $L$-vertices. For a given instance of Min-CVCB problem, let $f(k_u + k_l)$ be the number of leaves in the search tree, if in a step we can break the original problem into two sub-problems, and in each sub-problem the parameter scale can reduce $a$ and $b$ respectively, then we would establish a recurrence relation $f(k_u+k_l) \leq f(k_u+k_l-a)+f(k_u+k_l-b)$. When constructing search tree, we could include some blocks' $U$-part or $L$-part into $K$, until in a certain step breaks DAG D's NP-Hard structure, then uses dynamic programming technology to solve the surplus partial in the polynomial time.

In order to speed up the searching process, we will applythe technology of chain implication[9] , which makes full use of the block's adjacency relations to speed up the searching process significantly. Let $[B'_1, B'_2, \ldots, B'_h]$ be a path in the DAG D. If we include the $L$-part of the block $B'_1$ in $K$, then the $U$-part of the $B'_1$ must be excluded from $K$. Since there is an edge in $G$ from the $U$-part of $B'_1$ to $L$-part of the block $B'_2$, we must also include the $L$-part of the block $B'_2$ in $K$, which, in consequence, will imply that the $L$-part of the block $B'_3$ must be in $K$, and so on. In particular, the $L$-part of the block $B'_1$ in $K$ implies that the $L$-parts of all blocks $B'_2, \ldots, B'_h$ on the path must be in $K$. Similarly, the $U$-part of the block $B'_h$ in $K$ implies that $U$-parts of all blocks $B'_1, .., B'_{h-1}$ must be in $K$. This technology enables us to handle many cases very efficiently.

The particular operation of the algorithm is to list all the possible adjacency of the blocks in which we branch in the search process. First we analysis the corresponding branching of the blocks whose weight is no less than 4, then analysis all the possible joint of block whose weight is 3 to establish the searching tree. For the block whose weight is 3, first listing the possible joint of the block in a case-by-case exhaustive manner, and then makes the best of bounded search-trees technology to construct new recurrence relations. Let $\lambda_{in}(B_i)$ be the in-degree of the block $B_i$, $\lambda_{out}(B_i)$ be the out-degree of $B_i$, $w(B_i)$ be the weight of $B_i$, and $w(P_{B_i})$ be the weight of all the blocks that have a directed path to the block $B_i$. We would divide it into two situations as follows according to the block $B_0$'s weight.

1. $w(B_0) \geq 4$. Since the constrained minimum vertex cover $K$ of the DAG D either contains the entire $U_i$-part and is disjoint from the $L_i$-part, or contains the entire $L_i$-part and is disjoint from the $U_i$-part of the block $B_i$, we branch in this case by either including the entire $U_i$-part in $K$ (and remove the $L_i$-part from the graph) or including the entire $L_i$-part in $K$ (and removing the $U_i$-part from the graph). In each case, we add at least 4 vertices in $K$ and remove block $B_0$ from DAG D. Thus, this branch satisfies the recurrence relation

$$f(k_u + k_l) \leq 2f(k_u + k_l - 4) \tag{1}$$

2. $w(B_0) = 3$. According to the value of in-degree and out-degree of block $B_0$, we would divide it into four situations as follows.

2.1 $\lambda_{in}(B_0) \geq 1$ and $\lambda_{out}(B_0) \geq 1$. If we include the $U$-part of $B_0$ in $K$, it forces at least $3 + \lambda_{in}(B_0)$ vertices in $K$ by the chain implication. If we include the $L$-part of $B_0$ in $K$, it also forces at least $3 + \lambda_{out}(B_0)$ vertices in $K$. Thus in this case, the branching satisfies recurrence relation (1).

2.2 $\lambda_{out}(B_0) \geq 1$ and $\lambda_{in}(B_0) = 0$. According to the out-degree of $B_0$ and $w(P_{B_0})$, we would divide it into three situations as follows.

2.2.1 $w(P_{B_0}) \geq 3$. If we included the $U$-part of $B_0$ in $K$, it forces at least 3 vertices in $K$ by the "chain implication". If we include the $L$-part of $B_0$ in $K$, it forces at least 6 vertices in $K$. Thus, this branching satisfies the recurrence relation

$$f(k_u + k_l) \leq f(k_u + k_l - 3) + f(k_u + k_l - 6) \tag{2}$$

2.2.2 $w(P_{B_0}) = 2$. In this case, all the connections of block $B_0$ have eight cases shown in Fig.1, after excluding $B_0$ 's isolated connections. From Fig.1(a) to Fig.1(g), let the two connected blocks of $B_0$ be $B_1$ and $B_2$. When $\lambda_{in}(B_1) \geq 2$, the connected blocks is $B_3$, when $\lambda_{in}(B_2) \geq 2$, the connected blocks is $B_4$. In Fig.1(h), let the blocks connected with $B_1$ is $B_3$. We'll give an analysis of how to establish a bounded search tree in the following.

2.2.2.1 In Fig.1(a), $B_0$ is connected with two connected blocks $B_1$ and $B_2$ whose weight are 1, and $\lambda_{in}(B_1) \geq 2$, $\lambda_{in}(B_2) \geq 3$. When $w(B_3) > 1$, the time complexity of the branching is lower than the one when $w(B_3) = 1$, so we only need to consider the situation when $w(B_3) = 1$. It is also the same in the following context. In general, we only have to analyze the equal situation. When the situation of $\lambda_{in}(B_1) \geq 1$ is analyzed, the time complexity of branching is also lower than the situation when $\lambda_{in}(B_1) = 1$. Also, in the following, if it is required to analyze the in-degree or out-degree of a block whether the value is larger or equal to a constant, we only have to analyze the equal situation is enough.

Let the block $B_1$ be the core of branching: if the $U$-part of block $B_1$ is in $K$, it can be concluded by the chain implication that: the $U$-part of the block $B_0$ and $B_3$ are also in $K$, thus it equals that 5 vertices are included in the $K$. If the $L$-part vertices of block $B_1$ are in $K$, the block $B_0$ and $B_2$ become "isolated block". Thus, it equals that 5 vertices are included in the $K$. So, the branching is at least (5, 5), and the corresponding recurrence is just as formula

$$f(k_u + k_l) \leq 2f(k_u + k_l - 5) \tag{3}$$

2.2.2.2 In Fig.1(b), $B_0$ is connected with two connected blocks $B_1$ and $B_2$ whose weight are 1, and $\lambda_{in}(B_1) \geq 2$,$\lambda_{in}(B_2) = 2$.

Let $B_2$ be the core of branching, the problem under this situation is exactly the same as the (2.2.2.1), so the analysis is identical, and it can be branched at least (6, 5), and the corresponding recurrence is just as formula

$$f(k_u + k_l) \leq f(k_u + k_l - 6) + f(k_u + k_l - 5) \tag{4}$$

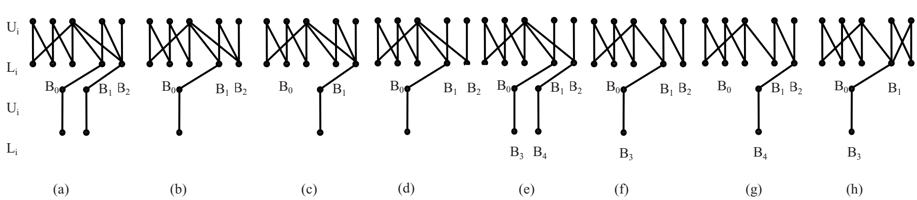


**Fig. 1.** All possible connections in DAG D when $w(P_{B_0}) = 2$

2.2.2.3 In Fig.1(c), $B_0$ is connected with two connected blocks $B_1$ and $B_2$ whose weight are 1, and $\lambda_{in}(B_1) = 1$,$\lambda_{in}(B_2) \geq 3$.

Let $B_2$ be the core of branching, the problem under this situation is exactly the same as the (2.2.2.1), so the analysis is identical, so it can be branched at least (6, 5), and the corresponding recurrence is just as formula (4).

2.2.2.4 In Fig.1(d), $B_0$ is connected with two blocks $B_1$ and $B_2$ with no connections whose weight are 1, and $\lambda_{in}(B_1) \geq 2$, $\lambda_{in}(B_2) = 1$.

Let $B_1$ be the core of branching, if $U$-part of block $B_1$ is in $K$, it can be concluded from the chain implication that: the $U$-part of the block $B_0$ and $B_4$ are also in $K$. thus, the $K$ contains at least 5 vertices and the block $B_2$ becomes the "isolated block". Thus it equals that 6 vertices are included in the $K$. If the $L$-part vertices of block $B_1$ are in $K$, the block $B_0$ and $B_2$ become "isolated block". Thus, it means that 5 vertices are included in the $K$. So the branching is at least (6, 5), and the corresponding recurrence is just as formula (4).

2.2.2.5 In Fig.1(e), $B_0$ is connected with two blocks $B_1$ and $B_2$ whose weight are 1 with no connections, and $\lambda_{in}(B_1) \geq 2$, $\lambda_{in}(B_2) \geq 2$.

From the "vertex folding" in Ref.[11], to contain the edges among the blocks $B_0, B_1, B_2, B_3, B_4$, one is to make the $U$-part vertices be included in $K$, which is equal to putting at least 4 vertices into $K$(corresponding to the situation that $B_3$and $B_4$ are in the same block, and Fig.1(de) gives an exact connection); the other is to make the L-part vertices be included in $K$, and it will make the block $B_0$ become the "isolated block", thus, it equals to includes at least 5 vertices into $K$. so branch is (4, 5), and the corresponding recurrence relation is formula

$$f(k_u + k_l) \leq f(k_u + k_l - 4) + f(k_u + k_l - 5) \tag{5}$$

2.2.2.6 In Fig.1(f), $B_0$ is connected with two blocks $B_1$ and $B_2$ which has no connections, and $\lambda_{in}(B_1) \geq 2$, $\lambda_{in}(B_2) = 2$.

Let $B_1$ be the core of branching, if the $U$-part of block $B_1$ are in $K$, it can be concluded by the chain implication that: the $U$-part of the block $B_0$ and $B_2$are also in $K$. Thus, the $K$ contains at least 5 vertices and the block $B_2$ becomes the "isolated block". Thus it equals that 6 vertices are included in the $K$. If the L-part vertices of block $B_1$ are in $K$, it can be contained at least 2 vertices in $K$ and $B_0$ becomes "isolated block". Thus it equals that 5 vertices are included in $K$. So it can be branched at least (6, 5), and the corresponding recurrence is just as formula (4).

2.2.2.7 In Fig.1(g), $B_0$ is connected with two blocks $B_1$ and $B_2$ whose weight is 1 and has no connections, and $\lambda_{in}(B_1) = 1$, $\lambda_{in}(B_2) \geq 2$.

To make the block $B_2$ as the core of branching, the problem under this situation is exactly the same as the (2.2.2.6), so the analysis is identical, so the branching is at least (6, 5), and the corresponding recurrence is just as formula (4).

2.2.2.8 In Fig.1(h), $B_0$ is connected with a blocks $B_1$ whose weight is 2, and $\lambda_{in}(B_1) \geq 2$.

Let $B_1$ be the core of branching, the problem under this situation is exactly the same as the (2.2.2.6), so the analysis is identical, and the branching is at least (6, 5), the corresponding recurrence is just as formula (4).

2.2.3 $w(P_{B_0}) = 1$

From $w(P_{B_0}) = 1$, we know that $B_0$ is connected with a blocks $B_1$ whose weight is 1, let another block that connects with $B_1$ is $B_2$ (when $\lambda_{in}(B_1)$, block $B_0$, $B_1$ become "isolated block"). Let $B_1$ be the core of branching, the problem under this situation is exactly the same as (2.2.2.6), so the analysis is identical, and the branching is at least (4, 5), the corresponding recurrence is just as formula (5).

2.3 $\lambda_{in}(B_0) \geq 1$ and $\lambda_{out}(B_0) = 0$

Under this situation, all kinds of connection in the DAG D is entirely symmetry like (2.2), so the handling method is just the same and we can get the same recurrence relation.

2.4 $\lambda_{in}(B_0) = 0$ and $\lambda_{out}(B_0) = 0$

The block $B_0$ becomes the "isolated block" and we can make full use of the dynamic programming technology to solve it in polynomial time in the fourth part.

Considering all the recurrence relations above, it is obvious that formula (1) is the strictest one. So a theorem can be presented as follows:

**Theorem 1.** *When a block $B_0$ in the DAG D satisfies the inequality $w(B_0) \geq 3$, the branching recurrence relation brought out by the branching process at least satisfies the formula $f(k_u + k_l) \leq 2f(k_u + k_l - 4)$.*

## 4 Algorithm EACI-dyn

After processing the blocks of weight larger than 3, the remain DAG $D$ contains only isolated blocks of weight 3 and connected subgraphs that are composed by blocks of weight 1 or 2. We can solve the Min-CVCB problem on this structure by dynamic programming. The corresponding algorithm is EACI-dyn. Let the connected subgraphs in the remaining DAG $D$ be $G'_i$, $1 \leq i \leq r$ ($r$ be the number of the connected subgraphs). Let $G_0 = G'_1 + G'_2 + \ldots + G'_r$, and let the number of vertices in the connected subgraph $G_i$ be $2n_i$. Therefore, the total number of vertices in the graph $G_0$ is $2n_0 = 2n_1 + \cdots + 2n_r$. We show that all the possible minimum vertex covers in each connected subgraph can be enumerated in polynomial time. Then the dynamic programming algorithm is used to find the minimum vertex cover in $G_0$ satisfying the constraints.

After enumerating all possible minimum vertex covers in each connected subgraph $G'_i$, the next step is to find a minimum vertex cover of size $(k_u, k_l)$ in the graph $G_0$. Obviously, $G_0$ has the minimum vertex cover of size $(k_u, k_l)$ if and only if each connected subgraph $G'_i$ has a minimum vertex cover of size $(k_u^{(i)}, k_l^{(i)})$, such that $k_u^{(1)} + \ldots + k_u^{(r)} \leq k_u$, and $k_l^{(1)} + \ldots + k_l^{(r)} \leq k_l$.

The procedure that finds a minimum vertex cover of size $(k_u, k_l)$ in the graph $G_0$ is as follows: let $\bar{c} = c_1 + \ldots + c_i$, $1 \leq i \leq r$, and $A[1 \ldots r, 0 \ldots k_u]$ be a matrix of size $r^*(k_u + 1)$. Each element $A[i, j]$ in the matrix is to record a minimum vertex cover of size $(j, \bar{c} - j)$ in the graph $G'_1 + \cdots + G'_i$. The matrix $A$ can be constructed by the dynamic programming algorithm in Fig. 2.

**Input**: the connected graphs of $G'_1, G'_2 \ldots G'_r$ after section 3's branching
**Output**: a minimum vertex cover $K$ of $G$ with at most $k_u$ $U$-vertices and at most $k_l$ $L$-vertices if such a minimum vertex cover exists

1. list all the possible minimum vertex cover of $G'_1, G'_2 \ldots G'_r$;
2. **foreach** $1 \leq i \leq r$,$0 \leq j \leq k_u$ **do**
    $A[i,j] = \phi$;
3. **foreach** $(k_u^{(1)}, k_l^{(1)})$*-minimum vertex cover of* $C'_1$ *of* $G'_1$ **do**
    $A[1, k_u^{(1)}] = C'_1$;
4. **for** $i = 1...r-1$ **do**
    **for** $j = 0...k_u$ **do**
        **if** $A[i,j] \neq \phi$ **then**
            let $[i,j] = V_u \cup V_l, V_u \subseteq U, V_l \subseteq L)$;
        **foreach** $(k_u^{(i+1)}, k_l^{(i+1)})$*-minimum vertex cover,*$C'_{i+1} = V_u^{(i+1)} \cup V_l^{(i+1)}$ *of* $G'_{i+1}$ *in the list* $L_{i+1}$ **do**
            $A[i+1, j+k_u^{(i+1)}] = (V_u \cup V_u^{(i+1)}) \cup (V_l \cup V_l^{(i+1)})$;
5. **for** $j = 0...k_u$ **do**
    **if** $(j \leq k_u)\&(n_0 - j \leq k_l)\&[r,j] \neq \phi)$ **then**
        then return $A[r,j]$;
6. **return** $\phi$;

**Fig. 2.** Algorithm. EACI-dyn.

**Theorem 2.** *The time complexity of the algorithm EACI-dyn is* $O((k_u+k_l)k_u^2)$.

*Proof.* After the branching process in section 3, the remaining DAG $D$ is composed of isolated blocks of weight 3 and blocks of weight 1 or 2. First, all possible minimum vertex covers of each connected subgraph $G'_i$, $1 \leq i \leq r$, can be listed in linear time, then the matrix $A$ can be constructed by the dynamic programming algorithm to find the constrained minimum vertex cover. In the dynamic programming algorithm, the number of the minimum vertex covers in every row $L_i$ of the matrix $A$ is at most $k_u$, and the value of the next row depends on the value of the above one, so the time complexity of constructing the matrix A is $O(rk_u^2)$, Since $r$ be the number of the connected subgraphs, and $r \leq (k_u + k_l)$, So, the running time of the algorithm EACI-dyn is bounded by $O((k_u + k_l)k_u^2)$.

## 5 Putting All Together

With all the previous discussions combined, an algorithm EACI is given in Fig.3, which solves the Min-CVCB problem. We explain the steps of the algorithm as follows.

Step 1 is the initialization of the vertex cover $K$. Steps 2 and 3 make immediate decisions on high-degree vertices. If a $U$-vertices $u$ of degree larger than $k_l$ is not in the minimum vertex cover $K$, then all neighbors of $u$ should be in $K$, which would exceed the bound $k_l$. Thus, every $U$-vertex of degree larger than $k_l$ should

**Input**: a bipartite graph $G = (U, L, E)$ and two integers $k_u$ and $k_l$
**Output**: a minimum vertex cover $K$ of $G$ with at most $k_u$ $U$-vertices and at most $k_l$ $L$-vertices, or report no such a vertex cover exists

1. $K = \phi$;
2. **foreach** *$U$-vertex $u$ of degree larger than $k_l$* **do**
   include $u$ in $K$ and remove $u$ from $G$; $k_u = k_u - 1$;
3. **foreach** *$L$-vertex $v$ of degree larger than $k_u$* **do**
   include $v$ in $K$ and remove $v$ from $G$; $k_l = k_l - 1$;
4. apply lemma 1 to reduce the instance so that $G$ is a bipartite graph with perfect matching and with at most $2(k_u + k_l)$ vertices (with the integers $k_u$ and $k_l$ and the minimum vertex cover $K$ also properly updated);
5. apply lemma 2 to decompose the graph $G$ into elementary blocks $B_1, B_2, \ldots, B_r$, sorted topologically;
6. for connections that contain the block $B_i$ in DAG D has weight at least 3, branching it according in section 3;
7. All other cases not in section3, we can use algorithm EACI-dyn to solve it in polynomial time in section 4;

**Fig. 3.** Algorithm. EACI.

be automatically included in $K$. Similar justification applies to $L$-vertices of degree larger than $k_u$. Of course, if $k_u$ or $k_l$ becomes negative in step 2 or step 3, then we should stop and claim the nonexistence of the desired minimum vertex cover. After these steps, the degree of the vertices in the graph is bounded by $k' = \max\{k_u, k_l\}$. Since now each vertex can cover at most $k'$ edges, the number of edges in the resulting graph must be bounded by $k'(k_u + k_l) \leq (k_u + k_l)^2$, otherwise the graph cannot have a minimum vertex cover of no more than $k_u + k_l$ vertices. In step 4, Lemma 1 allows us to further reduce the bipartite graph $G$ so that $G$ has a perfect matching (the integers $k_u$ and $k_l$ are also properly reduced). The number of vertices in the graph $G$ now is bounded by $2(k_u + k_l)$. Step 5 applies Lemma 2 to decompose the graph $G$ into blocks. Step 6 is to analyze all the possible minimum vertex covers on the condition that the weight of the blocks in the connected sub-graphs is no less than 3, then use "chain implication" and bounded search technology to reduce the searching space in order to construct the bounded-search tree. Step 7 further analyzes the possible minimum vertex cover of the connected sub-graphs after step 6, and then applies algorithm EACI-dyn to search for the constraint minimum vertex cover.

**Theorem 3.** *The algorithm EACI runs in time $O((k_u + k_l)|G| + 1.1892^{k_u+k_l})$, i.e, the Min-CVCB problem is solvable in time $O((k_u + k_l)|G| + 1.1892^{k_u+k_l})$.*

*Proof.* As explained above, the algorithm EACI solves the Min-CVCB problem correctly. Thus, we only need to verify the running time of the algorithm.

It is easy to verify that the total running time of steps 1-3 of the algorithm is bounded by $O((k_u + k_l)|G|)$. Step 4 applies Lemma 1 to further reduce the bipartite graph $G$, and the running time of this step is bounded by $(k_u + k_l)^3$

(note that in this step, the number $m$ of edges in the graph $G$ is bounded by $(k_u + k_l)^2$ and the number $n$ of vertices in the graph $G$ is bounded by $2(k_u + k_l)$). Step 5 applies Lemma 2 to decompose the graph $G$ into elementary bipartite subgraphs and it takes time $O(|E|^2)$. Since $|E|$ is the number of edges in $G$, and $|E| \leq (k_u + k_l)^2$, step 5 takes time $O((k_u + k_l)^4)$. In step 7, by Theorem 2, the running time of the algorithm EACI-dyn is bounded by $O((k_u + k_l)k_u^2)$.

The only place the algorithm EACI branches is in step 6. Let $f(k_u + k_l) = x^{k_u+k_l}$ be the function in Theorem 1. By Theorem 1, we have

$$f(k_u + k_l) \leq 2f(k_u + k_l - 4)$$

Solving this recurrence relation gives us $f(k_u + k_l) \leq 1.1892^{k_u+k_l}$. Combining all steps together, we derive that the running time of the algorithm EACI is bounded by $O((k_u+k_l)|G|)+(k_u+k_l)^3+(k_u+k_l)^4|+1.1892^{k_u+k_l}+(k_u+k_l)k_u^2) = O((k_u + k_l)|G| + 1.1892^{k_u+k_l})$, i.e., the Min-CVCB problem could be solved in $O((k_u + k_l)|G| + 1.1892^{k_u+k_l})$.

## 6 Conclusions

In this paper, we study the Min-CVCB problem that has important applications in the area of VLSI manufacturing. We develop an improved parameterized algorithm for the problem based on a deeper and more careful analysis on the structures of bipartite graphs. We propose new techniques to handle blocks of weight bounded by 3, and use new branch search technology to reduce searching space. Our improved algorithm is achieved by integrating these new techniques with the known techniques developed by other researchers. The running time of our algorithm is $O((k_u + k_l)|G| + 1.1892^{k_u+k_l})$, compared to the previous best algorithm for the problem of running time $O((k_u + k_l)|G| + 1.26^{k_u+k_l})$.

## References

1. Hasan, N., Liu, C.L.: Minimum Fault Coverage in Reconfigurable Arrays. In: Pro of the 18th International Symposium on Fault-Tolerant Computing (FTCS'88), pp. 348–353. IEEE Computer Society Press, Los Alamitos,CA (1988)
2. Niosson, N.J.: Principles of Artificial Intelligence. Tioga Publishing Co., Palo Alto, CA (1980)
3. Kuo, S.Y., Fuchs, W.: Efficient spare allocation for reconfigurable arrays. IEEE Des. Test 4, 24–31 (1987)
4. Blough, D.M., Pelc, A.: Complexity of fault diagnosis in comparison models. IEEE Trans.Comput. 41(3), 318–323 (1992)
5. Low, C.P., Leong, H.W.: A new class of efficient algorithms for reconfiguration of memery arrays. IEEE Trans. Comput. 45(1), 614–618 (1996)
6. Smith, M.D., Mazumder, P.: Generation of minimal vertex cover for row/column allocation in self-repairable arrays. IEEE Trans.Comput. 45, 109–115 (1996)
7. Downey, R., Fellows, M.: Parameterized Complexity. Springer, New York (1999)

8. Fernau, H., Niedermeier, R.: An efficient exact algorithm for constraint bipartite vertex cover. J. Algorithms 38, 374–410 (2001)
9. Chen, J., Kanj, I.A.: Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. Journal of Computer and System Science 67, 833–847 (2003)
10. Lovasz, L., Plummer, M.D.: Matching Theory. In: Annals of Discrete Mathematics, vol. 29, North-Holland, Amsterdam (1986)
11. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. Journal of Algorithms, 280–301 (2001)