Indian J Pure Appl Math (2026) 57:425–433
https://doi.org/10.1007/s13226-024-00561-z

ORIGINAL RESEARCH

More efficient algorithms for searching for several edges in a hypergraph

Ting Chen

Received: 15 April 2023 / Accepted: 6 February 2024 / Published online: 4 March 2024

Abstract The edge searching problem is a generalization of the classical group testing problem. Chen and Hwang studied the problem of searching for many edges in a hypergraph with rank r. They provided a competitive algorithm to identify all d defective edges in a hypergraph with d unknown. Recently, Hwang first gave a competitive algorithm to find all defective edges in a graph. Chen proposed a revised algorithm for the same problem requiring at most $d\lceil\log_2|E|\rceil + d^2 + 3d + 1$ tests. In this paper, we will revisit the result proposed by Chen and give a more detailed analysis which implies that the revised algorithm actually requires at most $d\lceil\log_2|E|\rceil + 5d + 1$ tests. Then we further study the edge searching problem in a hypergraph of rank r. Considering the special case of $r = 3$, we will present more efficient algorithms to identify all defective edges in hypergraphs of rank 3.

Keywords Edge searching problem · Competitive algorithm · Graph · Hypergraph

1 Introduction

In this paper, we study the edge searching problem on hypergraphs which is a generalization of classical group testing problem. Let $G = (V,\ E)$ be a hypergraph of $rank\ r$, i.e., $|e| \le r$ for all edges $e \in E$. In particular, if every edge has r vertices, then G is an $r\text{-}graph$. We also let $G(S)$ denote the subgraph of G induced by the set S of vertices. We want to identify a subset $D \subseteq E$, called the *defective set*, with a minimum number of *edge tests*, where an edge test takes an arbitrary subset S of V and asks whether the subgraph $G(S)$ contains a defective edge or not.

The edge testing problem is an extension of the group testing problem which has a much longer history ([6]). [2] first cast a special group testing problem as searching for a single edge in a complete bipartite graph. [1] was the first one who proposed the edge testing problem for general 2-graphs, thus bringing the "graph"into focus.

Let $|D| = d$ and denote by $M(G,\ d)$ the minimum number of edge tests guaranteeing to identify the all d defective edges in G. [1] conjectured for 2-graphs

$$M(G,\ 1) \le \lceil\log_2|E|\rceil + c, \quad \text{where } c \text{ is a constant.}$$

[5] proved the conjecture with $c = 1$ and showed that this result is tight for general G. [10] generalized it to hypergraphs with rank r by proving $M(G,\ 1) \le \lceil\log_2|E|\rceil + r - 1$.

Communicated by Jayanthan A V.

T. Chen (✉)
School of Statistics and Mathematics, Zhejiang Gongshang University, Hangzhou, People's Republic of China
E-mail: steafting@sina.com

In the $d > 1$ case, [6] conjectured for 2-graphs

$$M(G, d) \leq d\left(\lceil \log_2 \frac{|E|}{d} \rceil + c\right), \quad \text{where } c \text{ is a constant.}$$

[8] made a breakthrough into the $d > 1$ problem by proving this conjecture for $c = 7$.

For the case that d is unknown, [4] first attempted to solve the problem of searching for all defective edges in a hypergraph with rank r. They gave a competitive algorithm requiring $d\lceil \log_2 |E| \rceil + (r-1)^{\lfloor \frac{r}{2} \rfloor} d^r + o(d^r)$ tests, where $r \geq 2$ is assumed to be a small constant. The edge searching problem in hypergraphs has applications in biology ([9]).

Recently, [7] gave a competitive algorithm to find all defective edges in a 2-graph. [3] provided a revised algorithm for the same problem requiring at most $d\lceil \log_2 |E| \rceil + d^2 + 3d + 1$ tests. In this paper, we will revisit this result by making some revisions to the analysis of the algorithm. We will point out that the revised algorithm actually requires at most $d\lceil \log_2 |E| \rceil + 5d + 1$ tests.

Then we will further study the edge searching problem in hypergraphs with rank r. Considering the special case of $r = 3$, we will give a competitive algorithm requiring at most $d\lceil \log_2 |E| \rceil + 4d^2 + 8d + 1$ tests to identify all defective edges in a 3-graph. Then we will present a more efficient algorithm for hypergraphs of rank 3 with the result $d\lceil \log_2 |E| \rceil + 6d^2 + 8d + 1$, which is better than the result $d\lceil \log_2 |E| \rceil + 2d^3 + o(d^3)$ given by [4].

2 The analysis of the algorithm for 2-graphs

In the following, we will revisit the algorithm presented by [3] to find all d defective edges in a graph $G = (V, E)$ and give a new analysis of the result. First we describe the *halving procedure* as a subroutine of the algorithm. For a set A of n elements, the halving procedure tests a subset B of $\lceil \frac{n}{2} \rceil$ elements. If the outcome is positive, it iterates the procedure on B; else, it iterates the procedure on $A \setminus B$.

We also introduce the *TJ-procedure* in the algorithm. [8] commented that Triesch's procedure for $r = 2$, with a little modification, can be used to identify a single defective edge in G in $\lceil \log_2 |E| \rceil + 1$ tests even though G has many defective edges. This procedure is referred to as the TJ-procedure.

The following revised algorithm of identifying all d defective edges in a graph $G = (V, E)$ was given by [3]. **Algorithm**

The partition stage:

Step 1: Set $V_1 = V$, $V_2 = \cdots = V_d = \emptyset$, $I = \emptyset$ (I is the set of identified defective edges).

Step 2: Test V_1. If positive, then

- Use the TJ-procedure to identify a positive edge (v, u).
- Use the join subroutine to assign v to some V_i $(i > 1)$.
- Set $V_1 = V_1 \setminus \{v\}$, $V_i = V_i \cup \{v\}$ and $I = I \cup \{(v, u)\}$. If $|V_1| \geq 2$, go back to Step 2.

Step 3: If one of the V_j $(j > 1)$ is nonempty, we enter the search stage.

Step 4: Stop with no defective edge identified.

The join subroutine:

Suppose v is the vertex to be assigned.

Step 1: Set $i = 2$.

Step 2:
- If $(v, u) \in I$ for some $u \in V_i$, set $V_i' = \{u \in V_i : (v, u) \notin I\}$.
- Test $\{v\} \cup V_i'$. If positive, use the halving procedure to identify a defective edge (v, u).
- Set $I = I \cup \{(v, u)\}$, $i = i + 1$ and go back to Step 2.

Step 3: Add v to V_i.

The search stage:

Suppose the partition stage yields nonempty $V_1, \ldots, V_m$ for some $m \geq 2$.

Step 1: Set $i = 1$ and $j = 2$.

Step 2: Choose a vertex $v \in V_j$.

Step 3: Let $V_i(v) = \{u \in V_i : (v, u) \in E \setminus I\}$.

If $V_i(v) = \emptyset$, go to the next v and go back to Step 3.

If $V_i(v) \neq \emptyset$, test $\{v\} \cup V_i(v)$. If positive, use the halving procedure (with v attached

to every test) to identify a defective edge $(v,\ u)$. Set $I = I \cup \{(v,\ u)\}$ and go back to Step 3. If negative, go to the next v and go back to Step 3.
Do this for every $v \in V_j$. Then go to Step 4.

Step 4: Set $j = j + 1$. If $j \leq m$, go back to Step 2.
Step 5: Set $i = i + 1$. If $i < m$, set $j = i + 1$ and go back to Step 2.
Step 6: Stop.

Based on the above description of this algorithm, the author claimed the following result as Theorem 1 ([3]), which could be stated as follows.

Theorem 1 *([3])* Let $G(V,\ E)$ be an arbitrary graph which contains d defective edges. Then the revised algorithm identifies all defective edges of G with at most $d\lceil \log_2 |E| \rceil + d^2 + 3d + 1$ tests.

Now we show that the join subroutine of the partition stage may not be correct. In Step 2 of the join subroutine, when we test $\{v\} \cup V_i'$, if the outcome is negative, we should assign v to V_i and stop. However, if there exists some $u \in V_i$ such that $(v,\ u) \in I$ at the beginning of Step 2, then the partition stage yields V_i contains a defective edge $(v,\ u) \in I$. Thus a future test on the vertices of $\{w\} \cup V_i(w)$ in the search stage, where $w \in V_j$ (for some $j > i$), may encounter an identified defective edge $e = (v,\ u)$ and the procedure of this stage will identify e repeatedly.

We revise the join subroutine of the partition stage as follows.

The join subroutine

Suppose v is the vertex to be assigned.

Step 1: Set $i = 2$.
Step 2: • If $(v,\ u) \in I$ for some $u \in V_i$, set $i = i+1$ and go back to Step 2. Else set $V_i' = \{u \in V_i : (v,\ u) \in E\}$.
• Test $\{v\} \cup V_i'$. If positive, use the halving procedure to identify a defective edge $(v,\ u)$.
• Set $I = I \cup \{(v,\ u)\}$, $i = i + 1$ and go back to Step 2.
Step 3: Add v to V_i.

Now we also make some revisions to the analysis of Theorem 1 and show that it can be improved as follows.

Theorem 2 *Let $G(V,\ E)$ be an arbitrary graph which contains d defective edges. Then the algorithm with the revised join subroutine identifies all defective edges of G with at most $d\lceil \log_2 |E| \rceil + 5d + 1$ tests.*

Proof The algorithm consists of two stages, that are, the partition stage which contains a join subroutine and the search stage. In the partition stage with the revised join subroutine, V is partitioned into $V_1,\ V_2,\ \ldots,\ V_m$ such that no V_i contains a defective edge. The join subroutine adds a vertex v to a set V_i where $i = \min\{j \geq 1 \mid$ There is no defective edge in $\{v\} \cup V_j(v)\}$.

Then in the search stage, we test every single vertex v mixed with each single subset of partitioning. After a test with a positive outcome on the vertices of $\{v\} \cup V_i(v)$, the procedure of the search stage will identify a new defective edge by the halving procedure as long as such an edge exists. By examining all unidentified edges between different subsets V_i and V_j, the algorithm with the revised version identifies all defective edges in G.

Clearly, each defective edge is identified by either the TJ-procedure in $\lceil \log_2 |E| \rceil + 1$ tests, or the halving procedure in $\lceil \log_2 |E| \rceil$ tests. Then by including the possible positive test initiating the identification process, the number of tests consumed in each identification procedure is bounded by $\lceil \log_2 |E| \rceil + 2$. Thus the d defective edges cost a total number of at most $d(\lceil \log_2 |E| \rceil + 2)$ tests.

It remains to count negative tests occurred elsewhere except those in the TJ-procedure or the halving procedure. The partition stage stops with a negative test on the vertex set V_1. Each join subroutine ends with a negative test to add the vertex v. Since each v to be assigned corresponds to a distinct defective edge in D, there are at most $d + 1$ negative tests from the partition stage.

In the search stage, we count negative tests by distinguishing two cases. First, for each vertex v in $\bigcup_{j=2}^{m} V_j$, there exists at most one negative test on $\{v\} \cup V_1(v)$. Since each vertex $v \in \bigcup_{j=2}^{m} V_j$ corresponds to a distinct defective edge, there are at most d of them. Thus the number of negative tests we have to conduct over all such $\{v\} \cup V_1(v)$, where $v \in \bigcup_{j=2}^{m} V_j$, is bounded by d.

Second, we count negative tests that occur in the process by mixing vertices from V_k $(k \geq 2)$ and one vertex $v \in V_l$ $(l > k)$. According to the join subroutine, we know that the vertex v is joined in V_l because there is no defective edge in $\{v\} \cup V_l$. However, every vertex in V_l is connected by at least one defective edge to each set V_k

$(2 \le k < l)$. Say, for each set V_k $(2 \le k < l)$ and every vertex $v \in V_l$, such a combination of vertices by mixing V_k and $\{v\}$ corresponds to a distinct defective edge. Thus the number of negative tests we have to conduct over all such $\{v\} \cup V_k(v)$, where $v \in V_l$ and $k = 2, \ldots, l-1$, will also be bounded by d. Therefore, the upper bound for the number of negative tests consumed in the search stage is $d + d = 2d$ (not counting the negative tests in the halving procedure).

From the above discussion, we have the total number of tests in the algorithm is at most $d\left(\lceil \log_2 |E| \rceil + 2\right) + (d+1) + 2d = d\lceil \log_2 |E| \rceil + 5d + 1$, instead of the result as Chen described in Theorem 1. □

3 The main idea of the algorithm for 3-graphs

In the following, we will further study the edge searching problem in hypergraphs with rank 3. We follow Chen and Hwang's approach (2007) in general, but to make improvement, we have to resolve some problems unique to 3-graphs and competitive algorithms.

[4] commented that after a positive test, there is an efficient way to identify a defective edge. Similar to the case of 2-graphs, it was Johann who actually revealed in a private communication that Triesch's algorithm to find the single defective edge in general hypergraphs G with $\lceil \log_2 |E| \rceil + r - 1$ tests works with a little modification, even if G contains many defective edges. They referred to Johann's edge-testing version as the TJ-procedure.

Since it is important to us, we describe the main idea of the TJ-procedure here. Let $G = (V, E)$ be a hypergraph with rank 3 and $C = (v_1, \ldots, v_c)$ be a vertex cover on E. Then each edge contains a vertex of C. We say that a vertex $v_k \in C$ *leads* an edge e if e contains no $v_j \in C$ with $j < k$. Triesch's group tests are always on $\{v_1, \ldots, v_i\}$ for some i, which can be converted to edge tests on $V \setminus \{v_1, \ldots, v_i\}$. The testing ends when a leader of a defective edge is identified. Then by induction, the problem is reduced to finding a defective edge in a 2-graph. Thus Johann's edge-testing version of Triesch's algorithm can be used to identify one defective edge efficiently after a positive test in hypergraphs with rank 3.

Let CHT(3) denote our proposed algorithm for 3-graphs which consists of two stages, that are, the partition stage containing a subroutine JOIN(v) and the search stage. Similar to the revised algorithm for 2-graphs, the output when the partition stage is done will be just m nonempty subsets $V_1, V_2, \ldots, V_m$ of V such that none of the subgraphs $G(V_i)$ $(1 \le i \le m)$ contains a defective edge.

In the search stage of algorithm CHT(3), we need to identify defective edges whose vertices spread into different subsets. Since we study on 3-graphs, a defective edge may spread into two or three sets of $V_1, V_2, \ldots, V_m$. We have to take one or two vertices from outside to test together with vertices in V_i since such a combination of vertices may contain a defective edge. However, we have to avoid identifying a defective edge contained in such a combination which is already identified. Thus we need to further partition the subset V_i into $V_{i0}, V_{i1}, \ldots, V_{il}$ such that for each p $(0 \le p \le l)$, V_{ip} together with the outside one or two vertices does not contain an identified defective edge.

We introduce a new definition of *layered vertex cover* to obtain the subsets $V_{i0}, V_{i1}, \ldots, V_{il}$.

Definition 1 Given a set E_1 of edges (not necessarily of the same rank) in a hypergraph G. Construct a vertex cover $C_1 = (v_1, v_2, \ldots, v_c)$ of E_1 with $d(v_1) \ge d(v_2) \ge \cdots \ge d(v_c)$ where $d(v_i)$ is the degree of the vertex v_i in the subgraph obtained from E_1 by deleting $v_1, \ldots, v_{i-1}$ and their edges. If C_1 also contains a set $E_2 \subseteq E_1$ of edges, then construct a vertex cover C_2 of $G(C_1)$ by the same manner as constructing the vertex cover C_1 of E_1. If C_2 also contains a set E_3 of edges, then construct a vertex cover C_3 of $G(C_2)$ by the same manner as before, and so on. We finally obtain a sequence of vertex covers of $C_1, C_2, \ldots, C_l$ when C_l does not contain any edge of E_1. We call $L = (C_1, C_2, \ldots, C_l)$ a *layered vertex cover* of E_1.

For example, we are given a 3-graph $G = (V_1, E_1)$ with the vertex set $V_1 = \{1, 2, 3, 4, 5, 6, 7\}$ and the edge set $E_1 = \{\{1, 2, 3\}, \{1, 4, 5\}, \{1, 6, 7\}, \{2, 3, 4\}, \{2, 5, 6\}, \{3, 6, 7\}\}$. We construct a vertex cover $C_1 = (1, 2, 3)$ of E_1 with $d(1) \ge d(2) \ge d(3)$. Since C_1 also contains $\{1, 2, 3\} \in E_1$, we construct a vertex cover $C_2 = (1)$ of $G(C_1)$. Then we obtain a layered vertex cover $L = (C_1, C_2)$ where C_2 does not contain any edge of E_1.

Let W be the set of one or two vertices taken from outside of V_i. In order to partition the set V_i, we will construct a layered vertex cover of all identified defective edges in $W \cup V_i$ and denote it by $L(W \cup V_i)$. Here, we restrict that all the vertices of the layered vertex cover only come from the set V_i. In our application $|W| \le 2$, hence W doesn't contain a defective edge. We construct the layered vertex cover $L(W \cup V_i)$ to partition V_i into $V_{i0}, V_{i1}, \ldots, V_{il}$ such that $W \cup V_{ip}$ $(0 \le p \le l)$ does not contain an identified defective edge, and then test $W \cup V_{ip}$ further to identify all other defective edges in $W \cup V_i$.

As the former papers which deal with the $d > 1$ case, we will bound the number of tests in identifying each defective edge, including negative tests occurred during the process. Then the analysis of algorithm CHT(3) is reduced to counting negative tests elsewhere.

4 The algorithm for 3-graphs

Given a 3-graph $G = (V,\ E)$. Let I denote the set of currently identified defective edges in our algorithm. For a set S of vertices, let $I(S)$ denote the subset of I formed by those edges whose vertices all belong to S. In the following, we give an algorithm CHT(3) to identify all defective edges in a 3-graph G.

Algorithm CHT(3)

The partition stage:

Step 1. Set $V_1 = V$, $V_2 = \cdots = V_n = \emptyset$, and $I = \emptyset$.

Step 2. Test V_1. If positive, use the TJ-procedure to identify a defective edge $e = \{v_1, v_2,\ v_3\} \subseteq V_1$. Set $I = I \cup \{e\}$, and $V_1 = V_1 \setminus \{v_1\}$. Use the subroutine JOIN(v_1) to assign v_1 to some V_i $(i > 1)$. If $|V_1| \geq 3$, go back to Step 2.

Step 3. Stop.

The subroutine JOIN (v):

Suppose v is the vertex to be assigned.

Step 1. Set $i = 2$.

Step 2. If $I(V_i \cup \{v\}) \neq \emptyset$, set $i = i + 1$ and go back to Step 2. Else, set $V_i' = \{u \in V_i$: there exists $w \in V_i \setminus \{u\},$ such that $\{v,\ u,\ w\} \in E\}$.

Step 3. If $|V_i'| \geq 2$, test $\{v\} \cup V_i'$. If positive, use the TJ-procedure to identify a defective edge e. Set $I = I \cup \{e\}$, $i = i + 1$ and go back to Step 2.

Step 4. Assign v to V_i and stop.

Suppose $V_1,\ V_2,\ \ldots,\ V_m (m \leq n)$ are nonempty. If $m \geq 2$, go to the search stage.

The search stage:

Step 1. Set $i = 1$ and $j = 2$.

Step 2. Take a vertex $v \in V_j$. Let $V_i(v) = \{u \in V_i$: there exists $w \in V_i \setminus \{u\},$ such that$\{v,\ u,\ w\} \in E\}$. Say, $V_i(v)$ is the set of endpoints of the edges in $V_i \cup \{v\}$ excluding v itself.

Step 2.1 Construct a layered vertex cover $L(\{v\}) = (C_1,\ C_2,\ \ldots,\ C_l)$ of the identified defective edge set $I(V_i(v) \cup \{v\})$. By denoting $C_0 = V_i(v)$, we set $V_{i,\,s-1} = C_{s-1} \setminus C_s$ for $1 \leq s \leq l$ and $V_{il} = C_l$.

Step 2.2 (i) Set $k = 0$.
(ii) If $|V_{ik}| \geq 2$, test $\{v\} \cup V_{ik}$. If positive, use the TJ-procedure to identify one defective edge $e = \{v,\ a,\ b\}$. Set $I = I \cup \{e\}$ and reconstruct the layered vertex cover of $I(\{v\} \cup V_{ik} \cup \cdots \cup V_{il})$. Reset V_{ik}, $V_{i,\,k+1}$, …, $V_{il'}$ and set $l = l'$. If negative, set $k = k + 1$.
(iii) If $k \leq l$, go back to (ii).

Step 2.3 (i) Set $p = 1$.
(ii) Take a vertex $u \in V_{ip}$.
(iii) Set $q = 0$.
(iv) Test $\{u,\ v\} \cup V_{iq}$ (if for some $w \in V_{iq}$, $\{u,\ v,\ w\} \in I$, delete w temporarily from V_{iq} before this test). If positive, use the halving procedure (with $\{u,\ v\}$ attached to every test) on V_{iq} to identify a defective edge $e = \{u,\ v,\ x\}$ where $x \in V_{iq}$. Set $I = I \cup \{e\}$. If negative, set $q = q + 1$.
(v) If $q < p$, go back to (iv).
(vi) When $u \in V_{ip}$ is exhausted, set $p = p + 1$. If $p \leq l$, go back to (ii).

When $v \in V_j$ is exhausted, go to Step 3.

Step 3. Take two vertices $v_1,\ v_2$ from $V(i + 1,\ j)$ containing at least one vertex of V_j, where $V(i + 1,\ j) = \bigcup_{k=i+1}^{j} V_k$. Let $V_i(v_1,\ v_2) = \{v \in V_i : \{v_1,\ v_2,\ v\} \in E \setminus I\}$. Say, $V_i(v_1,\ v_2)$ is the set of endpoints of the edges in $V_i \cup \{v_1,\ v_2\}$ excluding v_1 and v_2.

Step 3.1 Construct a layered vertex cover $L(\{v_1,\ v_2\}) = (C_1,\ C_2,\ \ldots,\ C_h)$ of $I(V_i(v_1,\ v_2) \cup \{v_1,\ v_2\})$. By denoting $C_0 = V_i(v_1,\ v_2)$, set $V_{i,\,s-1} = C_{s-1} \setminus C_s$ for $1 \le s \le h$ and $V_{ih} = C_h$.

Step 3.2 (i) Set $t = 0$.
(ii) Test $\{v_1,\ v_2\} \cup V_{it}$ (if for some $z \in V_{it}$, $\{v_1,\ v_2,\ z\} \in I$, delete z temporarily from V_{it} before this test). If positive, use the the halving procedure (with $\{v_1,\ v_2\}$ attached to every test) on V_{it} to identify a defective edge $e = \{v_1,\ v_2,\ y\}$ where $y \in V_{it}$. Set $I = I \cup \{e\}$. If negative, set $t = t + 1$.
(iii) If $t \le h$, go back to (ii).

When all $v_1,\ v_2 \in V(i+1,\ j)$ are exhausted, go to Step 4.

Step 4. Set $j = j + 1$. If $j \le m$, go back to Step 2.

Step 5. Set $i = i + 1$. If $i < m$, set $j = i + 1$ and go back to Step 2.

Step 6. Stop.

Clearly, in the partition stage, V is partitioned into $V_1,\ V_2,\ \ldots,\ V_m$ such that no V_i contains a defective edge. Then in the search stage, we test each single subset of the partitioning together with the outside one or two vertices to search for a defective edge containing 3 vertices. After a test with a positive outcome on the vertices of $\{v\} \cup V_{ik}$ or $\{v_1,\ v_2\} \cup V_{it}$, the procedure of the search stage will identify a new defective edge as long as such an edge exists. By examining all unidentified edges in the subsets $V_i \cup V_j$ and $V_i \cup V(i+1,\ j)$ where $1 \le i < j \le m$, the algorithm CHT(3) will identify all defective edges in G.

It remains to count the number of tests the algorithm CHT(3) uses. Notice that each defective edge is identified by either the TJ-procedure in $\lceil \log_2 |E| \rceil + 2$ tests, or the halving procedure in $\lceil \log_2 |E| \rceil$ tests. Thus the number of tests consumed in each identification procedure is bounded by $\lceil \log_2 |E| \rceil + 3$, including the possible positive test initiating the identification process, and all negative tests occurred during that process. Therefore the d defective edges cost a total number of at most $d(\lceil \log_2 |E| \rceil + 3)$ tests. So it suffices to count the number N of negative tests occurred elsewhere in CHT(3).

Lemma 1 $N \le 4d^2 + 5d + 1$.

Proof We count the negative tests occurred elsewhere except those in the TJ-procedure or the halving procedure. The partition stage stops with a negative test on the vertex set V_1. Each subroutine JOIN(v) ends with a negative test to assign the vertex v. Since each v to be assigned corresponds to a distinct defective edge in D, the partition stage costs at most $d + 1$ negative tests.

In the search stage, all negative tests counted in N occur in Step 2 and Step 3. We first consider negative tests in Step 2.2 and Step 2.3. For the fixed index i and $v \in V_j$ taken in Step 2, let D_{vij} be the set of defective edges in $G(\{v\} \cup V_i)$ and $d_{vij} = |D_{vij}|$. Since each vertex in the set $V_{i1} \cup V_{i2} \cup \cdots \cup V_{il}$ corresponds to a distinct defective edge, for each $v \in V_j$ and V_i $(i < j)$ the number of negative tests in Step 2.2 is at most $l + 1 \le d_{vij} + 1$.

In Step 2.3, for each $u \in V_{ip}$ and V_{iq} $(q < p)$, there is at least one defective edge which contains u and v. Then each process of Step 2.3 uses at most d_{vij} negative tests. Therefore, for each fixed V_i and $v \in V_j$ $(i < j)$ in Step 2, the number of negative tests is at most $d_{vij} + 1 + d_{vij} = 2d_{vij} + 1$. By letting D_{ij} be the set of all defective edges in $G(V_i \cup V_j)$ and $d_{ij} = |D_{ij}|$, summing up all $v \in V_j$, j and i yields the following upper bound for the number of negative tests in Step 2:

$$\begin{aligned}
\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} (2d_{vij} + 1) &= \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} 2d_{vij} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} 1 \\
&= 2 \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} d_{vij} + \left(\sum_{j=2}^{m} \sum_{v \in V_j} 1 + \sum_{i=2}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} 1 \right) \\
&\le 2 \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} d_{ij} + \sum_{j=2}^{m} \sum_{v \in V_j} 1 + \sum_{i=2}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} 1 \\
&\le 2d + \sum_{j=2}^{m} \sum_{v \in V_j} 1 + \sum_{i=2}^{m-1} \sum_{j=i+1}^{m} \sum_{v \in V_j} 1.
\end{aligned}$$

Notice that each vertex $v \in \bigcup_{j=2}^{m} V_j$ corresponds to a distinct defective edge, so $\sum_{j=2}^{m} \sum_{v \in V_j} 1 \le d$ holds. For $2 \le i < j \le m$, every vertex in V_j is connected by at least one defective edge to each set V_i. Thus,

$\sum_{i=2}^{m-1}\sum_{j=i+1}^{m}\sum_{v\in V_j} 1 \le d$ holds. Therefore, we have

$$\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v\in V_j}(2d_{vij}+1) \le 2d+d+d = 4d.$$

Secondly, we count all the negative tests in Step 3. For $\{v_1,\ v_2\}$ taken from $V(i+1,\ j)$, similar as the former argument of Step 2, each process of Step 3.2 uses at most $h+1 \le d_{v_1ij}+d_{v_2ij}+1$ negative tests. Since $\{v_1,\ v_2\}$ contains at least one vertex of V_j, without loss of generality, we assume that $v_1 \in V_j$. Then by letting $\widehat{D_{ij}}$ be the set of all defective edges in $G(V_i \cup V_{i+1}), \ldots, G(V_i \cup V_{j-1})$ and $G(V_i \cup V_j)$ and denoting $\widehat{d_{ij}} = |\widehat{D_{ij}}|$, summing up all v_1, v_2 and $i,\ j$ yields the following upper bound for the number of negative tests in Step 3:

$$\begin{aligned}
&\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\ j)}(d_{v_1ij}+d_{v_2ij}+1)\\
&=\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\ j)} d_{v_1ij} + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\ j)} d_{v_2ij}\\
&\quad+\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\ j)} 1\\
&\le \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j} d\cdot d_{v_1ij} + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\widehat{d_{ij}} + \sum_{j=2}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(2,\ j)} 1\\
&\quad+\sum_{i=2}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\ j)} 1\\
&\le d\cdot\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} d_{ij} + \sum_{i=1}^{m-1}\left(\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\right)\widehat{d_{im}} + \sum_{j=2}^{m}\sum_{v_1\in V_j} d\\
&\quad+\sum_{i=2}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j} d\\
&\le d\cdot d + \sum_{i=1}^{m-1} d\cdot\widehat{d_{im}} + d\sum_{j=2}^{m}\sum_{v_1\in V_j} 1 + d\sum_{i=2}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j} 1\\
&\le d^2 + d\cdot d + d\cdot d + d\cdot d\\
&= 4d^2.
\end{aligned}$$

From the above discussion, the total number of negative tests N in algorithm CHT(3) is at most $(d+1)+4d+4d^2 = 4d^2+5d+1$. □

Theorem 3 *Let $G(V,\ E)$ be an arbitrary 3-graph which contains d defective edges. Then the algorithm* CHT(3) *identifies all defective edges of G with at most $d\lceil\log_2|E|\rceil + 4d^2 + 8d + 1$ tests.*

Proof Since the d defective edges cost a total number of at most $d(\lceil\log_2|E|\rceil+3)$ tests and Lemma 1 shows that $N \le 4d^2+5d+1$, the total number of tests in algorithm CHT(3) is at most $d(\lceil\log_2|E|\rceil+3)+4d^2+5d+1 = d\lceil\log_2|E|\rceil+4d^2+8d+1$. □

5 The algorithm for hypergraphs with rank 3

In the following, let $G(V, E)$ be a hypergraph of rank 3, i.e., $|e| \le 3$ for all edges $e \in E$. It is assumed that no defective edge is contained in another. We follow the general approach in algorithm CHT(3) with a slight modification and denote CHT*(3) the algorithm for hypergraphs of rank 3.

The partition stage in CHT*(3) is same as that in CHT(3) except the differences of the rank of each edge and dealing with defective edges of rank 1. When a defective edge of rank 1 is identified in the partition stage, it is deleted from G. Thus when the partition stage is done, the output $V_1, V_2, \ldots, V_m$ still satisfies that each subgraph $G(V_i)$ $(1 \le i \le m)$ contains no defective edge.

The search stage in CHT*(3) will be a little different from CHT(3). When we take two vertices $\{v_1,\ v_2\}$ in Step 3, before constructing a layered vertex cover $L(\{v_1,\ v_2\})$ and then testing a subset V_{it} together with $\{v_1,\ v_2\}$, we will test $\{v_1,\ v_2\}$ itself. If the outcome is positive, then $\{v_1,\ v_2\} \in D$. By our assumption, there is no other defective edge containing $\{v_1,\ v_2\}$, so we do not need to test $\{v_1,\ v_2\} \cup V_{it}$ further. If the outcome is negative, we still need to construct $L(\{v_1,\ v_2\})$ and test $\{v_1,\ v_2\} \cup V_{it}$ to identify all defective edges containing $\{v_1,\ v_2\}$.

Algorithm CHT *(3)

The partition stage:

Step 1. Set $V_1 = V$, $V_2 = \cdots = V_n = \emptyset$, and $I = \emptyset$.
Step 2. Test V_1. If positive, use the TJ-procedure to identify a defective edge $e \subseteq V_1$. Set $I = I \cup \{e\}$. If $e = \{v\}$, delete it from G. Else, use the subroutine JOIN*(v) to assign one vertex $v \in e$ to some V_i $(i > 1)$. Set $V_1 = V_1 \setminus \{v\}$ and go back to Step 2.
Step 3. Stop.

The subroutine JOIN *(v):

Suppose v is the vertex to be assigned.

Step 1. Set $i = 2$.
Step 2. If $I(V_i \cup \{v\}) \neq \emptyset$, set $i = i + 1$ and go back to Step 2. Else, set $V_i' = \{u \in V_i : \{v, u\} \in E$ or there exists $w \in V_i \setminus \{u\}$, such that $\{v, u, w\} \in E\}$. Say, V_i' is the set of endpoints of the edges in $V_i \cup \{v\}$ excluding v itself.
Step 3. If $|V_i'| \ge 1$, test $\{v\} \cup V_i'$. If positive, use the TJ-procedure to identify a defective edge e. Set $I = I \cup \{e\}$, $i = i + 1$ and go back to Step 2.
Step 4. Assign v to V_i and stop.

The search stage:

We omit most of the steps in the search stage of CHT*(3) which are similar to those of CHT(3). In the search stage of CHT*(3), Step 3 with some modifications is described here.

Step 3. Take $\{v_1,\ v_2\} \notin I$ from $V(i+1,\ j)$ containing at least one vertex of V_j, where $V(i+1,\ j) = \bigcup_{k=i+1}^{j} V_k$.
  Step 3.1 If v_1, v_2 spread into different subsets of $V(i+1,\ j)$, test $\{v_1,\ v_2\}$. If positive, set $I = I \cup \{\{v_1,\ v_2\}\}$ and go back to Step 3.
  Step 3.2 Let $V_i(v_1,\ v_2) = \{v \in V_i : \{v_1,\ v_2,\ v\} \in E \setminus I\}$. Construct a layered vertex cover $L(\{v_1,\ v_2\}) = (C_1,\ C_2,\ \ldots,\ C_h)$ of $I(V_i(v_1,\ v_2) \cup \{v_1,\ v_2\})$. By denoting $C_0 = V_i(v_1,\ v_2)$, set $V_{i,\,s-1} = C_{s-1} \setminus C_s$ for $1 \le s \le h$ and $V_{ih} = C_h$.
  Step 3.3 Test $\{v_1,\ v_2\} \cup V_{it}$ for $0 \le t \le h$ to identify all defective edges in $G(\{v_1,\ v_2\} \cup V_i)$. The three sub-steps (i), (ii) and (iii) are completely same as Step 3.2 in algorithm CHT(3).

When all $v_1,\ v_2 \in V(i+1,\ j)$ are exhausted, go to the next step.

Similar to the argument of the algorithm CHT(3), we can show that the algorithm CHT*(3) identifies all defective edges in a hypergraph of rank 3. The analysis of the tests number of CHT*(3) is also similar to that of CHT(3). The number of tests consumed in the identification procedure of each defective edge is bounded by $\lceil \log_2 |E| \rceil + 3$ as well. Therefore the d defective edges cost a total number of at most $d(\lceil \log_2 |E| \rceil + 3)$ tests. We count the number of negative tests N^* occurred elsewhere in CHT*(3).

Lemma 2 $N^* \le 6d^2 + 5d + 1$.

Proof The upper bounds for the number of negative tests in the partition stage and Step 2 in the search stage of CHT*(3) are same as those in CHT(3), that are, $d + 1$ and $4d$ respectively.

The only difference between N^* and N is the number of negative tests in Step 3 of the search stage. For $v_1,\ v_2 \in V(i+1,\ j)$ taken in CHT*(3), each process of Step 3 uses at most $(h+1)+1 \le (d_{v_1ij} + d_{v_2ij} + 1) + 1$ negative tests instead of $d_{v_1ij} + d_{v_2ij} + 1$ negative tests in CHT(3). Then we have the following upper bound for

the number of negative tests in Step 3:

$$\begin{aligned}
&\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\,j)}(d_{v_1ij}+d_{v_2ij}+2)\\
&=\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\,j)}(d_{v_1ij}+d_{v_2ij}+1)+\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\,j)}1\\
&\le 4d^2+\sum_{j=2}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(2,\,j)}1+\sum_{i=2}^{m-1}\sum_{j=i+1}^{m}\sum_{v_1\in V_j}\sum_{v_2\in V(i+1,\,j)}1\\
&\le 4d^2+d^2+d^2=6d^2.
\end{aligned}$$

From above, the total number of negative tests N^* in algorithm CHT*(3) is at most $(d+1)+4d+6d^2=6d^2+5d+1$. □

Consequently, we have

Theorem 4 *Let $G(V,\ E)$ be an arbitrary hypergraph of rank 3 which contains d defective edges, where d is unknown. Then the algorithm* CHT*(3) *identifies all defective edges of G with at most $d\lceil\log_2|E|\rceil+6d^2+8d+1$ tests.*

Acknowledgements We thank the reviewers for their efforts to improve the readability of this paper.

References

1. Aigner, M. (1988). Combinatorial Search. In: Wiley-Teubner Series in Computer Science. Wiley. New York.
2. Chang, G. J., & Hwang, F. K. (1981). A group testing problem on two disjoint sets. SIAM J. Algebraic Discrete Methods, 2, 35-38.
3. Chen, T. (2011). A revised algorithm for searching for all defective edges in a graph. Discrete Applied Mathematics, 159, 2266-2268.
4. Chen, T., & Hwang, F. K. (2007). A competitive algorithm in searching for many edges in a hypergraph. Discrete Applied Mathematics, 155, 566-571.
5. Damaschke, P. (1994). A tight upper bound for group testing in graphs. Discrete Applied Mathematics, 48, 101-109.
6. Du, D. Z., & Hwang, F. K. (1993). Combinatorial Group Testing and its Applications. World Scientific. Singapore.
7. Hwang, F. K. (2005). A competitive algorithm to find all defective edges in a graph. Discrete Applied Mathematics, 148, 273-277.
8. Johann, P. (2002). A group testing problem for graphs with several defective edges. Discrete Applied Mathematics, 117, 99-108.
9. Torney, D. C. (1999). Set pooling designs. Annals of Combinatorics, 3, 95-101.
10. Triesh, E. (1996). A group testing problem for hypergraphs of bounded rank. Discrete Applied Mathematics, 66, 185-188.