entropy

MDPI

Article

Nonasymptotic Upper Bounds on Binary Single Deletion Codes via Mixed Integer Linear Programming

Albert No

Department of Electronic and Electrical Engineering, Hongik University, Seoul 04066, Korea; albertno@hongik.ac.kr; Tel.: +82-2-320-1649

Received: 11 November 2019; Accepted: 5 December 2019; Published: 6 December 2019

Abstract: The size of the largest binary single deletion code has been unknown for more than 50 years. It is known that Varshamov–Tenengolts (VT) code is an optimum single deletion code for block length $n \leq 10$; however, only a few upper bounds of the size of single deletion code are proposed for larger n. We provide improved upper bounds using Mixed Integer Linear Programming (MILP) relaxation technique. Especially, we show the size of single deletion code is smaller than or equal to 173 when the block length n is 11. In the second half of the paper, we propose a conjecture that is equivalent to the long-lasting conjecture that "VT code is optimum for all n". This equivalent formulation of the conjecture contains small sub-problems that can be numerically verified. We provide numerical results that support the conjecture.

Keywords: deletion channel; maximum independent set; mixed integer programming; Varshamov–Tenengolts code

1. Introduction

A deletion channel is one of the most important channels in the history of communication. The channel has a deletion error where the symbol is being removed without knowing the position of it. Unlike many other channels where the positions of symbols remain the same, the decoder needs to specify the position of each symbol, which is called a synchronization issue. Mainly due to this issue, the deletion channel is surprisingly hard to analyze.

There are several different mathematical problem formulations of the deletion channel. One natural way is the probabilistic approach where the deletion occurs in i.i.d. manner with probability p. Kanoria and Montanari provided an approximation of the channel capacity of binary deletion channel when $p \to 0$ [1]. However, the channel capacity is unknown in this setting even in binary case.

Alternatively, we can define the problem in algebraic way. We assume that there will be at most k deletion errors while transmitting n number of symbols. The question is the maximum size of the deletion code that can correct any k deletion errors. There is a nice survey paper by Sloane [2], and it is easy to see that the problem is extremely challenging even in single deletion case.

Although the problem is still open, there has been some progress in this algebraic setting. In binary case, Varshamov and Tenengolts proposed a simple code (VT-code) construction that corrects any single deletion error [3]. It is asymptotically optimum when n grows while the number of deletions $k = 1$ is fixed. VT-code is known to be optimum when $n \leq 10$ and conjectured that it is optimum for all n. Tenengolts generalized VT-code to a non-binary version [4]. Gabrys and Sala proposed a code that can correct two deletions [5]. Sima and Bruck also generalized VT-code that can correct k deletions [6]. Both results [5,6] show $n - \log |\mathcal{C}| = O(k \log n)$ where $\mathcal{C}$ is the deletion code. This implies

that they achieved order optimal redundancy where the optimality is shown by Levenshtein [7]. However, their works do not specify the coefficient of $k \log n$ term while VT code satisfies $n - \log |\mathcal{C}| = \log n + o(\log n)$ in the single deletion case.

VT-code naturally suggests a lower bound of the size of the optimum single deletion code in binary case, but the upper bounds are rarely provided. Levenshtein found an analytic formula for upper bounds [8]. Kulkarni and Kiyavashi proposed better upper bounds for any number of deletions and any size of alphabet [9]. Cullina and Kiyavashi refined Levenshtein's upper bound [10].

There are attempts to find nonasymptotic upper bounds numerically for small n. The most popular approach is using graph theory, which is based on the fact that the optimum single deletion code problem has an equivalent formulation of maximum independent set problem. In this formulation, the size of the largest single deletion code is equal to the size of the maximum independent set. A well-known upper bound of the maximum independent set is Lovász theta number, which can be obtained by solving Semidefinite Programming (SDP) relaxed problem [11]. Upper bounds from Lovász theta number are tight for $n \leq 8$; in other words, VT-code is optimum when the block length n is smaller than 8. Butenko et al. provided a numerical approach to bound the size of maximum independent set [12], and showed that VT-code is optimum when the block length is $n = 9$. Kulkarni et al. proposed a bounding technique using Linear Programming (LP) relaxation of the graph problem [13]. This bound is weaker than that of Lovász's, but the complexity is lower so that we can compute the bounds for larger block lengths n. For recent progress, we refer to Sloane's webpage [14], which also mentions that VT-code is optimum when the block length is $n = 10$.

In this paper, we propose a new method that provides an improved nonasymptotic upper bound. We partially relax the graph problem, and obtain Mixed Integer Linear Program (MILP) where some of variables are relaxed but some of them are not. MILP has reasonably low complexity and provides a better bound. For example, when the block length is $n = 11$, we show the size of optimum single deletion code is less than or equal to 173, where the SDP provides a bound of 174.

We also find an equivalent formulation of the conjecture that "VT-code is optimum". This equivalent form consists of several small sub-conjectures where VT-code is optimum if and only if all those sub-conjectures are true. The advantage of this equivalent conjecture is that we can numerically verify the sub-conjectures using (Mixed) Integer Programming. Note that we can disprove the original conjecture if any of sub-conjectures are not true. We numerically verified some of sub-conjectures for $n \leq 16$. This does not prove or disprove the optimality of VT-code, but it supports the original conjecture.

The remainder of the paper is organized as follows. In Section 2, we revisit some of the known results of single deletion codes. Section 3 presents the new bounding technique using Mixed Integer Linear Programming with improved upper bound. In Section 4, we present an equivalent formula of the conjecture as well as some numerical supports. Finally, we conclude in Section 5.

Notation

Let $\mathcal{X} = \{0, 1\}$ be a set of binary alphabet. We denote a n-tuple using super script, i.e., $x^n = x_1 x_2 \cdots x_n \in \mathcal{X}^n$. In addition, let 0^n be an all zero vector. For clarity, we use $x^n \in \mathcal{X}^n$ for an n-dimensional binary vector, and $y^{n+1} \in \mathcal{X}^{n+1}$ for an $n+1$-dimensional binary vector. We also use concatenation of binary vectors. For example, $x^n 0$ is an $n+1$-dimensional binary vector where the last bit is 0.

2. Preliminaries

2.1. Single Deletion Code

Define a deletion ball $\mathcal{B}_D(x^n) \subset \mathcal{X}^{n-1}$, which is a collection of $(n-1)$-dimensional binary vectors that are one-bit deleted version of x^n. We can define an insertion ball $\mathcal{B}_I(x^n) \subset \mathcal{X}^{n+1}$ in a similar manner.

Definition 1. *For a positive integer n, a set of binary vectors $\mathcal{C} \subset \mathcal{X}^n$ is a single deletion code if $\mathcal{B}_D(c_1) \cap \mathcal{B}_D(c_2) = \emptyset$ for all $c_1 \neq c_2$ in $\mathcal{C}$.*

The definition implies that the single deletion code can always correct a single deletion error. The following lemma shows that the single deletion code can also correct a single insertion error.

Lemma 1. *([13] Lemma 2.1) A set of binary vectors $\mathcal{C} \subset \mathcal{X}^n$ is a single deletion code if and only if $\mathcal{B}_I(c_1) \cap \mathcal{B}_I(c_2) = \emptyset$ for all $c_1 \neq c_2$ in $\mathcal{C}$.*

The above lemma is simply from the fact that $\mathcal{B}_I(c_1) \cap \mathcal{B}_I(c_2) \neq \emptyset$ if and only if $\mathcal{B}_D(c_1) \cap \mathcal{B}_D(c_2) \neq \emptyset$

2.2. Varshamov–Tenengolts Codes

Let $v_n : \mathcal{X}^n \to \mathbb{Z}$ be a function that computes the "VT-weights" of an n-dimensional binary vector.

$$v_n(x^n) = \sum_{k=1}^{n} k \cdot x_k.$$

Note that we do not take any modulo operations, and therefore $v_n(x^n)$ can take value from 0 to $\frac{n(n+1)}{2}$.

For $0 \leq a \leq n$, VT-code [3] is defined by

$$VT_a(n) = \{x^n \in \mathcal{X}^n : v_n(x^n) \equiv a \pmod{n+1}\}$$

which is a single deletion code [7,15]. Levenshtein showed that $VT_a(n)$ is perfect for all $0 \leq a \leq n$ [16]. In other words,

$$\mathcal{X}^{n-1} = \bigcup_{x^n \in VT_a(n)} \mathcal{B}_D(x^n).$$

For any a, we have $|VT_0(n)| \geq |VT_a(n)| \geq |VT_1(n)|$ where the first inequality is from Varshamov [3] and the second inequality is from Ginzburg [17]. Thus, the size of the optimum single deletion code is lower bounded by $|VT_0(n)|$. An analytic formula $|VT_0(n)|$ is given in [2]:

$$|VT_0(n)| = \frac{1}{2(n+1)} \sum_{\text{odd } d|n+1} \phi(d) 2^{(n+1)/d}$$

where $\phi(d)$ is Euler's totient function.

Borchers showed that $VT_0(n)$ is optimum single deletion code for $n \leq 10$ [14]. The optimality of VT-code is still open for $n \geq 11$. In the case of $n = 11$, the size of VT-code is $|VT_0(n)| = 172$ but the best known upper bound of the largest single deletion code is $|\mathcal{C}| \leq 174$.

2.3. Maximum Independent Set Approach

Consider a graph where all binary vectors are nodes. There exists an edge between two nodes x^n and $\tilde{x}^n$ if and only if $\mathcal{B}_D(x^n) \cap \mathcal{B}_D(\tilde{x}^n) \neq \emptyset$. Then, the optimum single deletion code corresponds to the maximum independent set. Note that the Maximum Independent Set (MIS) problem is NP-complete.

There are reduction rules in graph theory that provide an equivalent graph problem while reducing the size of the graph. Isolated vertex removal technique is useful in our case. An isolated vertex is a node for which its neighborhood forms a clique. For example, in our case, the neighborhood set of node 0^n is $\{100\cdots00, 010\cdots00, \ldots, 00\cdots01\}$, and $\{00\cdots00, 100\cdots00, 010\cdots00, \ldots, 00\cdots01\}$ forms a clique. This implies that the node 0^n is an isolated vertex. Butenko et al. showed that there

exists a maximum independent set that contains all isolated vertices [12]. Thus, there exists an optimum single deletion code that contains both 0^n and 1^n.

Note that it is still infeasible to solve our MIS problem with state of the art algorithms [18,19] when $n \geq 11$. Segundo et al. proposed a variation of BBMC [20] and found the maximum independent set of the graph induced from two deletion ($k = 2$) channel [14]. However, the graph induced from single deletion channel ($k = 1$) is more challenging to find the maximum independent set.

2.4. LP Relaxation

For simplicity, we define several functions and new notations. Let $N = 2^n$, and $[N-1] = \{0, 1, \ldots, N-1\}$ be the set that contains all nonnegative integers smaller than or equal to $N-1$. We further let $b_n : [N-1] \to \{0,1\}^n$ be the function that converts the decimal number to the binary vector (e.g., $b_4(3) = 0011$). Note that we drop n if it is clear from the context, i.e., $b \equiv b_n$.

In the above section, we define a graph where there exists an edge between x^n and $\tilde{x}^n$ if their deletion balls share an element. Instead, we define an equivalent graph where the set of nodes are $V = [N-1]$. The set of edges $E \subset V \times V$ is derived naturally where $(i,j) \in E$ if and only if there is an edge between $b(i)$ and $b(j)$.

Our goal is to find an independent set $\mathcal{U} \subset V$ that has maximum number of elements. For $0 \leq i \leq N-1$, let $X = (X_0, X_1, \ldots, X_{N-1})$ be binary variables where $X_i = 1$ if $i \in \mathcal{U}$ and $X_i = 0$ if $i \notin \mathcal{U}$. If there exists an edge between i and j, then the independent set $\mathcal{U}$ cannot contain both i and j. Thus, the following Integer Programming (IP) problem is equivalent to the maximum independent set problem.

$$\begin{aligned} \max_{X} \quad & \sum_{i=0}^{N-1} X_i \\ \text{s.t.} \quad & X_i + X_j \leq 1, \quad \text{for } (i,j) \in E \\ & X_i \in \{0,1\}, \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

Clearly, it is an NP-hard problem, which is extremely challenging to solve. Instead, we can relax it to an easier problem, and bounding the solution of the IP. One way of doing it is classical Linear Programming (LP) relaxation, which is given by

$$\begin{aligned} \max_{X} \quad & \sum_{i=0}^{N-1} X_i \\ \text{s.t.} \quad & X_i + X_j \leq 1, \quad \text{for } (i,j) \in E \\ & 0 \leq X_i \leq 1, \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

LP relaxation allows the variable X_i to take a value between 0 and 1, and the solution of relaxed problem provides an upper bound of the original IP. However, this gives a trivial solution in our case, which is $X_i = 1/2$ for all $i \in \{0, 1, \ldots N-1\}$. The maximum value of the objective function is 2^{n-1}, and it is much larger than the known upper bound $\frac{2^n-2}{n-1}$ [9].

Kulkarni et al. proposed another LP relaxation [9]. The idea is that the independent set can contain at most one node from each clique. For any $(n-1)$-dimensional vector y^{n-1}, the set of nodes $\mathcal{Q}_I(y^{n-1}) \triangleq \{i : b(i) \in \mathcal{B}_I(y^{n-1})\}$ forms a clique since $y^{n-1} \in \mathcal{B}_D(b(i)) \cup \mathcal{B}_D(b(j))$ for all $i, j \in \mathcal{Q}_I(y^{n-1})$. Thus, the independent set $\mathcal{U}$ can take at most one node from $\mathcal{Q}_I(y^{n-1})$, and we have *clique constraints* $\sum_{i \in \mathcal{Q}_I(y^{n-1})} X_i \leq 1$. This implies another LP relaxation which provides a tighter upper bound of the original IP problem.

$$\begin{aligned} \max_{X} \quad & \sum_{i=0}^{N-1} X_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{Q}_I(y^{n-1})} X_i \leq 1, \qquad \text{for } y^{n-1} \in \mathcal{X}^{n-1} \\ & 0 \leq X_i \leq 1, \qquad i = 0, 1, \ldots, N-1. \end{aligned}$$

On the other hand, Lovász proposed an SDP-relaxation of the Maximum Independent Set (MIS) problem [11]. The solution of SDP problem is called Lovász theta number, which provides a tighter bound of MIS problem.

The following table presents the upper bound from the above relaxations as well as $|VT_0(n)|$ which is a lower bound.

n	6	7	8	9	10	11	12
$\lvert VT_0(n)\rvert$	10	16	30	52	94	172	316
Lovász	10.00	16.84	30.00	53.03	95.98	174.73	-
LP	10.25	17.15	30.32	53.56	96.52	175.19	321.27

Note that the complexity of LP relaxed problem is low, and we can get bounds for $n \geq 13$ as well. For example, the size of maximum deletion code is smaller than or equal to 593 when $n = 13$. However, due to the complexity issue, we are not able to compute Lovász theta number for $n \geq 12$. For example, it took more than 24 h on our machine when $n = 12$.

3. Mixed Integer Linear Programming

LP is faster than the original IP problem; however it is hard to parallelize. On the other hand, IP inherently uses branch-and-bound technique which can be parallelized, but still intractable with current multi-thread processors. In this section, we propose a Mixed Integer Linear Programming (MILP) problem, which is in between LP problem and the original IP problem.

3.1. Main Results

In the LP relaxation, all variables are relaxed as described in Section 2.4. Instead, we relax specific variables only, which provides a semi-relaxed optimization problem. More precisely, we design $\mathcal{S} \subset V$ and keep X_i to be binary variable for $i \in \mathcal{S}$ while other variables are relaxed as in LP relaxation. This provides a Mixed Integer Programming (MIP) problem where variables are either integer or real numbers:

$$\begin{aligned} \max_{X} \quad & \sum_{i=0}^{N-1} X_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{Q}_I(y^{n-1})} X_i \leq 1, \qquad \text{for } y^{n-1} \in \mathcal{X}^{n-1} \\ & X_i \in \{0, 1\}, \qquad \text{for } i \in \mathcal{S} \\ & 0 \leq X_i \leq 1, \qquad \text{for } i \notin \mathcal{S} \\ & X_0 = 1 \\ & X_{N-1} = 1. \end{aligned}$$

The last two constraints are because there exists a maximum independent set that contains both 0^n and 1^n from Section 2.3.

MIP is generally as computationally demanding as IP problems. However, since all constraints are linear, the above optimization problem is a Mixed Integer Linear Programming (MILP) problem. MILP can be solved in reasonable amount of time if we carefully design the set $\mathcal{S}$. Let $MILP_n(\mathcal{S})$ denote the above MILP problem.

If $\mathcal{S} = \emptyset$, then $MILP_n(\mathcal{S})$ is equivalent to LP. On the other extreme, if $\mathcal{S} = V$, then $MILP_n(\mathcal{S})$ is equivalent to the original IP. If $\mathcal{S}$ is nontrivial subset of $\mathcal{X}^n$, then the solution of $MILP_n(\mathcal{S})$ provides a tighter upper bound of maximum independent set problem while having low complexity.

Clearly, we prefer smaller $\mathcal{S}$ because of complexity. Thus, the goal is designing $\mathcal{S}$ in smart way. The main idea is increasing the size of $\mathcal{S}$ in greedy manner. We start from fully relaxed LP problem, i.e., $\mathcal{S} = \emptyset$, and add elements one by one under certain criterion.

More precisely, we solve $MILP_n(\mathcal{S})$ in each iteration and add a node i to $\mathcal{S}$ based on the following rule. Let $d : [N-1] \to \{0, 1, \ldots, 2^{n-1}\}$ be the function which indicates the number of *clique constraints* that contains i. Since $i \in \mathcal{Q}_I(y^{n-1})$ if and only if $y^{n-1} \in \mathcal{B}_D(b(i))$, we have $d(i) = |\mathcal{B}_D(b(i))|$. Thus, the variable X_i affects the $d(i)$ number of clique constraints. Furthermore, if X_i is large, it restricts other variables in *clique constraints* more. Thus, we measure the amount of "impact" of variable X_i by $d(i) \times X_i$. Finally, the algorithm finds the node i that maximizes $d(i) \times X_i$ and add it to $\mathcal{S}$. This procedure is described in Algorithm 1.

Algorithm 1 Sequential MILP.

Input: target threshold τ

Output: new bound T

procedure SEQMILP(τ)
  Set $\mathcal{S} = \emptyset$, and $T = \inf$
  do
    Solve $MILP_n(\mathcal{S})$ and let T be the objective function value and X be the solution
    $i_0 = \arg\max_{i \notin \mathcal{S}} d(i) \times X_i$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{i_0\}$
  while $|\mathcal{S}| < N$ and $T \geq \tau$
  return T
end procedure

The above algorithm takes a target threshold τ as an input which can be a previously known upper bound of the original IP. In each iteration, it computes the objective function value T of $MILP_n(\mathcal{S})$. Whenever T is smaller than the target bound τ, then the algorithm halts and we get a new bound T of the size of maximum independent set. For example, suppose we let $\tau = |VT_0(n)| + 1$ and the above program halts with $T < \tau$, then we have a new upper bound that the size of the maximum single deletion code is strictly smaller than $|VT_0(n)| + 1$. In such case, we can claim that $VT_0(n)$ is the optimum single deletion code. On the other hand, suppose the program ends with $|\mathcal{S}| = N$ which means $\mathcal{S} = V$. In such case, the return value T is the size of the maximum independent set because it is the objective function value of the original IP. Note that the size of $\mathcal{S}$ is increased by 1 in each iteration, and therefore there will be at most N iterations.

Note that the way of choosing $i_0 = \arg\max_{i \notin \mathcal{S}} d(i) \times X_i$ is not an optimum way. However, it is an effective way, as shown in Section 3.2.

3.2. Experiments

We implemented Python code using PULP python package [21] with cbc solver [22]. Note that cbc solver supports multiple threads. For our experiments, we used a machine with AMD Threadripper 1950X processor and 64 GB of RAM. The operating system was Ubuntu 18.04 LTS.

In the case of $n = 11$, for any single deletion code $\mathcal{C}$, we have an upper bound $|\mathcal{C}| \leq 174$ from Lovász theta number. LP provides a weaker upper bound $|\mathcal{C}| \leq 175$. However, the proposed MILP problem $MILP_n(\mathcal{S})$ provides $|\mathcal{C}| \leq 174.993$ when

$$\begin{aligned}\mathcal{S} = \{&216, 1702, 1236, 1140, 1678, 1050, 154, 105, 538,\\ &1701, 1941, 532, 907\}.\end{aligned}$$

Note that we convert the binary vectors to decimal numbers for simplicity (e.g., 00011011000 $\rightarrow$ 216). This implies $|\mathcal{C}| \leq 174$ which is the same upper bound from Lovász number, but obtained more efficiently.

More interestingly, $MILP_n(\mathcal{S})$ provides $|\mathcal{C}| \leq 173.988$ when

$$\begin{aligned}\mathcal{S} = \{&216, 1702, 1236, 1140, 1678, 1050, 154, 105, 538,\\ &1701, 1941, 532, 907, 1335, 1194, 1515, 1942, 1366,\\ &688, 1200, 1365, 682, 854, 778, 340, 576, 847,\\ &1851, 853, 1300, 537, 322, 70, 681, 556, 1359,\\ &1749, 196, 1755, 1707, 1709, 1533, 1371, 1539, 667,\\ &1306, 1750, 514, 1708, 1323, 811, 724, 1706, 756,\\ &603, 1440, 1878, 1370, 938, 1364, 683, 842, 378,\\ &1354, 1509, 1450, 1258, 850, 598, 1373, 1494, 1482,\\ &1362, 553, 554, 1492, 333, 1493, 674, 1498, 1386,\\ &874, 596, 1738, 1690, 426, 1378\}.\end{aligned}$$

Note that it took 27,096 s to solve the last (88-th) $MILP_n(\mathcal{S})$ with 28 threads. This provides an improved upper bound $|\mathcal{C}| \leq 173$. Thus, we have the following Corollary.

Corollary 1. *For $n = 11$, the size of single deletion code is smaller than or equal to 173.*

In case of $n = 12$, for any single deletion code $\mathcal{C}$, an upper bound from LP implies $|\mathcal{C}| \leq 321$. However, the proposed method $MILP_n(\mathcal{S})$ provides $|\mathcal{C}| < 320.998$ when

$$\begin{aligned}\mathcal{S} = \{&3627, 1215, 360, 45, 423, 4050, 2880, 3024, 1071,\\ &1047, 3048, 1625, 1958, 3714, 3906\}.\end{aligned}$$

Thus, we have an improved upper bound $|\mathcal{C}| \leq 320$.

3.3. Connection to Metaheuristics

Although MILP problem has lower complexity than the original IP problem, MILP often encounter the computational issue as well. This is because the most state-of-the-art MILP solvers such as CPLEX [23] are based on branch-and-bound techniques, and it often has exponentially large search space. Thus, the smartly fixing the variable to binary is necessary as we presented in the previous section.

Similar heuristic algorithms appear in various other computationally challenging (Mixed) Integer Programming problems, such as lot sizing problem [24,25] and connected facility location problem [26]. This is commonly referred to as hybrid metaheuristics. For example, Wilbaut and Hanafi proposed iterative idea to solve MIP problems [27]. The authors applied this idea to IP problems such as knapsack [28]. The idea is iteratively solving the LP relaxed problem to get an upper bound and reduced problem with fixing variables to get a lower bound until the lower and upper bounds match. In this paper, we do not fix the value of variable, but remove the relaxed constraints (so that some

variables remain binary). For other works in metaheuristics, we refer the interested reader to the nice survey paper by Blum et al. [29].

4. Equivalent Conjecture

The above semi-relaxation provides an improved upper bound; however, the running time is still an issue. In this section, we provide smaller optimization problems that can provide insights for the optimality of $VT_0(n)$ code.

4.1. VT-Sum Based Partition

For $0 \leq i \leq \frac{n(n+1)}{2}$, let $S_{n,i} \subset \mathcal{X}^n$ be the set of binary vectors whose VT-weights are i.

$$S_{n,i} = \{x^n \in \mathcal{X}^n : v_n(x^n) = i\}.$$

Clearly, $\{S_{n,i} : 0 \leq i \leq \frac{n(n+1)}{2}\}$ is a partition of $\mathcal{X}^n$, and $\{S_{n,i} : 0 \leq i \leq \frac{n(n+1)}{2},\ i \equiv a \pmod{n+1}\}$ is a partition of $VT_a(n)$.

The following lemma provides useful properties of $S_{n,i}$.

Lemma 2. *Suppose n and $0 \leq i, j \leq \frac{n(n+1)}{2}$ are positive integers. Then,*

1. $|S_{n,i}| = |S_{n,\frac{n(n+1)}{2}-i}|$.
2. *If $|i-j| \geq n+1$, then $\mathcal{B}_D(S_{n,i}) \cap \mathcal{B}_D(S_{n,j}) = \emptyset$.*
3. *If $|i-j| \geq n+1$, then $\mathcal{B}_I(S_{n,i}) \cap \mathcal{B}_I(S_{n,j}) = \emptyset$.*

Proof.

1. There exists a one-to-one correspondence between $S_{n,i}$ and $S_{n,\frac{n(n+1)}{2}-i}$ which is an element-wise binary complement.
2. For any element $z^{n-1} \in \mathcal{B}_D(S_{n,i})$, it is easy to show that $i - n \leq v_{n-1}(z^{n-1}) \leq i$. If $|i-j| \geq n+1$, then $\{i, i-1, \ldots, i-n\} \cap \{j, j-1, \ldots, j-n\} = \emptyset$, and therefore $\mathcal{B}_D(S_{n,i}) \cap \mathcal{B}_D(S_{n,j}) = \emptyset$.
3. For any $x^n, \tilde{x}^n \in \mathcal{X}^n$, it is clear that $\mathcal{B}_D(x^n) \cap \mathcal{B}_D(\tilde{x}^n) = \emptyset$ if and only if $\mathcal{B}_I(x^n) \cap \mathcal{B}_I(\tilde{x}^n) = \emptyset$. Thus, the third property is a direct consequence of the second property.

□

Remark 1. *If we view the original problem as a maximum independent set problem, the above partition $S_{n,0}, \ldots, S_{n,\frac{n(n+1)}{2}}$ can be useful since:*

- *There are no internal edges in $S_{n,i}$ for all i.*
- *There are no edges between $S_{n,i}$ and $S_{n,j}$ if $|i-j| \geq n+1$.*

This is not exactly a "partite" graph but has a similar flavor of it, and there might be an efficient way of finding a maximum independent set.

4.2. Equivalent Conjecture

For a given single deletion code $\mathcal{C}$, we also define a similar partition of $\mathcal{C}$. For $0 \leq i \leq \frac{n(n+1)}{2}$, we let $\mathcal{C}_i = \mathcal{C} \cap S_{n,i} = \{x^n \in \mathcal{C} : v_n(x^n) = i\}$. Then, we are ready to state our first lemma which is a building block of the main conjecture.

Lemma 3. *Let n be a positive integer, and $\mathcal{C} \subset \mathcal{X}^n$ be a single deletion code. If there exists an integer $0 \leq k \leq \lfloor \frac{n}{2} \rfloor$ such that*

$$|C_0| + |C_1| + \ldots + |C_{k(n+1)}| > |S_{n,0}| + |S_{n,n+1}| + \ldots + |S_{n,(n+1)k}|, \tag{1}$$

then $VT_0(n)$ is not an optimum single deletion code.

Proof. Suppose there exists k such that $|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k}| > |S_{n,0}| + |S_{n,n+1}| + \cdots + |S_{n,(n+1)k}|$. Then, we can define a new single deletion code

$$\tilde{\mathcal{C}} = \left(\bigcup_{i=0}^{(n+1)k} \mathcal{C}_i \right) \cup \left(\bigcup_{j=k+1}^{\lfloor n/2 \rfloor} S_{n,(n+1)j} \right).$$

Clearly, $\tilde{\mathcal{C}}$ is a single deletion code since $\mathcal{C}_i \cap S_{n,(n+1)j} = \emptyset$ for all $i \leq (n+1)k$ and $j \geq k+1$. Then, the size of the new single deletion code is given by

$$\begin{aligned} |\tilde{\mathcal{C}}| &= \sum_{i=0}^{(n+1)k} |\mathcal{C}_i| + \sum_{j=k+1}^{\lfloor n/2 \rfloor} |S_{n,(n+1)j}| \\ &> \sum_{j=0}^{k} |S_{n,(n+1)j}| + \sum_{j=k+1}^{\lfloor n/2 \rfloor} |S_{n,(n+1)j}| \\ &= \sum_{j=0}^{\lfloor n/2 \rfloor} |S_{n,(n+1)j}| \\ &= |VT_0(n)|. \end{aligned}$$

This concludes the proof. □

By the first property of Lemma 2, we have $|VT_0(n)| = |VT_{\frac{n+1}{2}}(n)|$ for odd n. Thus, we have the following lemma as well, which is essentially the same as Lemma 3.

Lemma 4. *Let n be an odd positive integer, and $\mathcal{C} \subset \mathcal{X}^n$ be a single deletion code. If there exists an integer $0 \leq k \leq \frac{n-1}{2}$ such that*

$$|C_0| + |C_1| + \ldots + |C_{k(n+1)+\frac{n+1}{2}}| > |S_{n,\frac{n+1}{2}}| + |S_{n,n+1+\frac{n+1}{2}}| + \ldots + |S_{n,(n+1)k+\frac{n+1}{2}}|, \tag{2}$$

then $VT_0(n)$ is not an optimum single deletion code.

Proof. Suppose n is odd and there exists k such that $|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k+\frac{n+1}{2}}| > |S_{n,\frac{n+1}{2}}| + |S_{n,n+1+\frac{n+1}{2}}| + \cdots + |S_{n,(n+1)k+\frac{n+1}{2}}|$. Then, we can define a new single deletion code

$$\tilde{\mathcal{C}} = \left(\bigcup_{i=0}^{(n+1)k+\frac{n+1}{2}} \mathcal{C}_i \right) \cup \left(\bigcup_{j=k+1}^{\frac{n-1}{2}} S_{n,(n+1)j+\frac{n+1}{2}} \right).$$

Again, the size of the new code is given by

$$\begin{aligned} |\tilde{\mathcal{C}}| &= |\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k+\frac{n+1}{2}}| + \sum_{j=k+1}^{\frac{n-1}{2}} |S_{n,(n+1)j+\frac{n+1}{2}}| \\ &> |S_{n,\frac{n+1}{2}}| + |S_{n,n+1+\frac{n+1}{2}}| + \cdots + |S_{n,(n+1)k+\frac{n+1}{2}}| \\ &= |S_{n,\frac{(n+1)(n-1)}{2}}| + |S_{n,\frac{(n+1)(n-3)}{2}}| + \cdots + |S_{n,0}| \\ &= |VT_0(n)|, \end{aligned}$$

This concludes the proof. □

Conjecture 1.

1. *For even n, any single deletion code $\mathcal{C} \subset \mathcal{X}^n$ satisfies*

$$|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k}| \leq |S_{n,0}| + |S_{n,n+1}| + \cdots + |S_{n,(n+1)k}|$$

   for all $0 \leq k \leq \frac{n}{2}$.
2. *For odd n, any single deletion code $\mathcal{C} \subset \mathcal{X}^n$ satisfies*

$$|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k+(n+1)/2}| \leq |S_{n,(n+1)/2}| + |S_{n,n+1+(n+1)/2}| + \cdots + |S_{n,(n+1)k+(n+1)/2}|$$

   for all $0 \leq k \leq \frac{n-1}{2}$.

The following theorem tells us that the above conjecture is equivalent to the original conjecture that "VT-code is optimum".

Theorem 1. *$VT_0(n)$ code is an optimum single deletion code if and only if Conjecture 1 holds.*

Proof. Note that the "only if" part (for both even and odd n) directly comes from Lemmas 3 and 4.

Suppose n is even and $|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k}| \leq |S_{n,0}| + |S_{n,n+1}| + \cdots + |S_{n,(n+1)k}|$ for all k. If we let k be $n/2$, then we have

$$\begin{aligned}|\mathcal{C}| =& |\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)n/2}| \\ \leq& |S_{n,0}| + |S_{n,n+1}| + \cdots + |S_{n,(n+1)n/2}| \\ =& |VT_0(n)|.\end{aligned}$$

Suppose n is odd and $|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)k+(n+1)/2}| \leq |S_{n,(n+1)/2}| + |S_{n,n+1+(n+1)/2}| + \cdots + |S_{n,(n+1)k+(n+1)/2}|$ for all k. If we let k be $(n-1)/2$, then we have

$$\begin{aligned}|\mathcal{C}| =& |\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{(n+1)n/2}| \\ \leq& |S_{n,(n+1)/2}| + |S_{n,n+1+(n+1)/2}| + \cdots + |S_{n,(n+1)n/2}| \\ \leq& |S_{n,(n+1)(n-1)/2}| + |S_{n,(n+1)(n-3)/2}| + \cdots + |S_{n,0}| \\ =& |VT_0(n)|.\end{aligned}$$

□

4.3. Special Case of $k = 1$

Theorem 1 implies that if we can find any code that satisfies the inequality in Equation (1) or the inequality in Equation (2), then VT code is not optimum. Thus, the plausible strategy to disprove the conjecture is finding a counterexample for Conjecture 1. However, in this section, we show the inequality in Equation (1) is true for all n when $k = 1$.

We define two functions that are useful in the remaining sections. The following lemma is for the definition of the first function.

Lemma 5. *For any n-dimensional binary vector x^n,*

$$v_{n+1}(\mathcal{B}_I(x^n)) \stackrel{\Delta}{=} \{v_{n+1}(\tilde{x}^{n+1}) : \tilde{x}^{n+1} \in \mathcal{B}_I(x^n)\} = \{v_n(x^n), v_n(x^n) + 1, \ldots, v_n(x^n) + n + 1\}.$$

Proof. It is well-known that the size of $\mathcal{B}_I(x^n)$ is $n+2$ for all x^n [30]. Clearly, a single insertion can only increase a VT-sum, in other words, for $y^{n+1} \in \mathcal{B}_I(x^n)$,

$$v_{n+1}(y^{n+1}) \geq v_n(x^n).$$

On the other hand, a single insertion can increase a VT-sum by at most $n+1$ by adding "1" to the last position. More precisely, for $y^{n+1} \in \mathcal{B}_I(x^n)$, we have

$$v_{n+1}(y^{n+1}) \leq v_n(x^n) + n + 1 = v_{n+1}(x^n 1).$$

Note that $\{v_{n+1}(y^{n+1}) : y^{n+1} \in \mathcal{B}_I(x^n)\}$ are all distinct, and therefore

$$v_{n+1}(\mathcal{B}_I(x^n)) = \{v_n(x^n), v_n(x^n)+1, \ldots, v_n(x^n)+n+1\}.$$

□

First, we define a map g which maps x^n to y^{n+1} where $v_{n+1}(y^{n+1}) \equiv 0 \bmod (n+1)$ and $y^{n+1} \in \mathcal{B}_I(x^n)$. More precisely, we have $g : \mathcal{X}^n \to \bigcup_{i\geq 0} S_{n+1,(n+1)i}$, where $g(x^n) \in \mathcal{B}_I(x^n)$. Since $v_{n+1}(\mathcal{B}_I(x^n)) = \{v_n(x^n), \ldots, v_n(x^n)+n+1\}$ consists of $n+2$ consecutive numbers, we can determine $g(x^n)$ uniquely when $x^n \notin VT_0(n)$. On the other hand, if $x^n \in VT_0(n)$, both $x^n 0$ and $x^n 1$ are possible candidates for $g(x^n)$. In this case, we set $g(x^n) = x^n 0$.

Define another function $h : \mathcal{X}^{n+1} \to \mathcal{X}^n$, where $h(x^{n+1}) = x^n$. The function h simply deletes the last bit. If we combine two functions, we get $f(x^n) = h(g(x^n))$. Then, the function f satisfies the following properties.

Lemma 6.

1. *For all $x^n \in \mathcal{X}^n$, we have $f(x^n) \in VT_0(n)$.*
2. *If $x^n \in VT_0(n)$, then $f(x^n) = x^n$.*
3. *For all $x^n \in \mathcal{X}^n$, we have $|v_n(f(x^n)) - v_n(x^n)| \leq n$.*

Proof.

1. Let $y^{n+1} = g(x^n)$, then we have

$$v_n(f(x^n)) \equiv v_n(h(y^{n+1})) \equiv v_{n+1}(y^{n+1}) \equiv 0 \pmod{n+1}.$$

2. If $v_n(x^n) \equiv 0 \bmod (n+1)$, then we have $g(x^n) = x^n 0$, and therefore $f(x^n) = x^n$.
3. If we let $y^{n+1} = g(x^n)$, then we have

$$\begin{aligned} v_n(f(x^n)) - v_n(x^n) =& v_n(y^n) - v_n(x^n) \\ =& v_{n+1}(y^{n+1}) - (n+1)y_{n+1} - v_n(x^n) \\ =& v_{n+1}(g(x^n)) - (n+1)y_{n+1} - v_n(x^n). \end{aligned}$$

Recall that we have $v_n(x^n) \leq v_{n+1}(y^{n+1}) \leq v_n(x^n) + n + 1$. If $v_n(x^n) \in VT_0(n)$, then $f(x^n) = x^n$, which implies $v_n(f(x^n)) - f_n(x^n) = 0$. On the other hand, if $v_n(x^n) \notin VT_0(n)$, we have $v_n(x^n) + 1 \leq v_{n+1}(g(x^n)) \leq v_n(x^n) + n$. In such case, $v_n(f(x^n)) - v_n(x^n)$ achieves the maximum value when $y_{n+1} = 0$ and $v_{n+1}(g(x^n)) = v_n(x^n) + n$. In other words,

$$v_{n+1}(g(x^n)) - v_n(x^n) - (n+1)y_{n+1} \leq v_n(x^n) + n - v_n(x^n) = n.$$

On the other hand, it achieves the minimum value when $y_{n+1} = 1$ and $v_{n+1}(g(x^n)) = v_n(x^n) + 1$. In other words,

$$v_{n+1}(g(x^n)) - v_n(x^n) - (n+1)y_{n+1} \geq v_n(x^n) + 1 - v_n(x^n) - (n+1) = -n.$$

Thus, we have

$$|v_n(f(x^n)) - v_n(x^n)| \leq n.$$

This concludes the proof.

□

Then, we have a theorem that supports Conjecture 1.

Theorem 2. *For positive integer n, any single deletion code $\mathcal{C}$ satisfies the following inequality.*

$$|\mathcal{C}_0| + |\mathcal{C}_1| + \cdots |\mathcal{C}_{n+1}| \leq |S_{n,0}| + |S_{n,n+1}|.$$

Proof. Let $\mathcal{C}' = \bigcup_{i=0}^{n+1} \mathcal{C}_i$. From the above lemma, we have $v_n(f(x^n)) \leq v_n(x^n) + n \leq 2n + 1$ for all $x^n \in \mathcal{C}'$. Since $f(x^n) \in VT_0(n)$, it is clear that $v_n(f(x^n))$ should be either 0 or $n + 1$. In other words,

$$f(x^n) \in S_{n,0} \cup S_{n,n+1}.$$

Suppose there exists $x^n, \tilde{x}^n \in \mathcal{C}'$ such that $f(x^n) = f(\tilde{x}^n) = y^n$, then $v_n(y^n)$ is either 0 or $n + 1$. First, if $v_n(y^n) = 0$, then $y^n = 0^n$. In this case, the weights (number of ones) of x^n and $\tilde{x}^n$ are at most one, i.e., $w(x^n), w(\tilde{x}^n) \leq 1$. This implies that $0^{n-1} \in \mathcal{B}_D(x^n) \cap \mathcal{B}_D(\tilde{x}^n)$ which is a contradiction. On the other hand, consider the case where $v_n(y^n) = n + 1$. Since $g(x^n), g(\tilde{x}^n) \leq 2n + 1$ and $v_{n+1}(y^n 1)$, we have $g(x^n) = g(\tilde{x}^n) = y^n 0$. This implies that $\mathcal{B}_I(x^n) \cap \mathcal{B}_I(\tilde{x}^n)$ has at least one element, and therefore $\mathcal{B}_D(x^n) \cap \mathcal{B}_D(\tilde{x}^n) \neq \emptyset$. This is a contradiction.

Thus, $f|_{\mathcal{C}'} : \mathcal{C}' \to S_{n,0} \cup S_{n,n+1}$ is an injective function, and therefore

$$|\mathcal{C}'| = |\mathcal{C}_0| + |\mathcal{C}_1| + \cdots + |\mathcal{C}_{n+1}| \leq |S_{n,0}| + |S_{n,n+1}|.$$

□

4.4. Integer Programming

As mentioned above, if we can find a single deletion code that satisfies the inequality in Equation (1) or the inequality in Equation (2), that immediately disproves the optimality of VT code. Since the size of the problem is relatively smaller when k is small, we can numerically solve the Integer Programming (IP) problem without relaxation.

For example, we can check whether the inequality in Equation (1) holds or not for some fixed n and k, using the following optimization problem.

$$\begin{aligned} \max_{X} \quad & \sum_{i:v_n(b(i)) \leq (n+1)k} X_i \\ \text{s.t.} \quad & \sum_{i \in \mathcal{Q}_I(y^{n-1})} X_i \leq 1, \quad \text{for } y^{n-1} \in \mathcal{X}^{n-1} \\ & X_i \in \{0,1\}, \quad i = 0, 1, \ldots, N-1. \end{aligned}$$

For some n and k, if the maximum value of the objective function is smaller than or equal to

$$M_{n,k} \triangleq |S_{n,0}| + |S_{n,n+1}| + \ldots + |S_{n,(n+1)k}|,$$

then it means that no single deletion code satisfies the inequality in Equation (1).

Note that the number of variables are $|\{x^n \in \mathcal{X}^n : v_n(x^n) \leq (n+1)k\}|$, which is strictly smaller than 2^n. We solve the above optimization numerically for various n and k. For all combinations that we tried, the maximum value of the objective function is $M_{n,k}$. The following table shows all combinations of (n, k) that we were able to check in a reasonable amount of time.

n	11	12	13	14	15	16
$k=1$	✓	✓	✓	✓	✓	✓
$k=2$	✓	✓	✓	✓	✓	✓
$k=3$	✓	✓	✓	✓	✓	✓
$k=4$	✓	✓	✓	✓		

Entries without check mark are combinations that we could not check due to the running time.

Similarly, we numerically verify the inequality in Equation (2) from Lemma 4. For all combinations of (n, k) that we tried, we were not able to find any counterexample. The following table shows all combinations that we were able to check.

n	11	13	15
$k=1$	✓	✓	✓
$k=2$	✓	✓	✓
$k=3$	✓	✓	✓

5. Conclusions

We investigated the maximum size of binary single deletion code. Tighter upper bounds are provided using Mixed Integer Linear Programming relaxation. In the case of $n = 11$, we showed that the size of the largest single deletion code is smaller than or equal to 173. This implies that the largest single deletion code is size of either 172 or 173. In addition, we showed a conjecture that is equivalent to the optimality of VT code. Numerical results are proposed that support the conjecture that VT code is an optimum single deletion code.

One possible direction for future work is semi-relaxed Semidefinite Programming problem. Since Lovász number (which is based on Semidefinite Programming) provides a better bound than LP, we can propose the Mixed Integer Semidefinite Programming as we semi-relaxed the LP problem. Solvers for Mixed Integer Semidefinite Programming are not as popular as solvers for MILP except a few initial works [31]. However, we think it can verify the optimality of VT code in the case of $n = 11$.

Funding: This work was supported by the Samsung Research Funding and Incubation Center for Future Technology under Grant SRFC-IT1802-09

Conflicts of Interest: The author declares no conflict of interest.

References

1. Kanoria, Y.; Montanari, A. Optimal coding for the binary deletion channel with small deletion probability. *IEEE Trans. Inf. Theory* **2013**, *59*, 6192–6219. [CrossRef]
2. Sloane, N.J. On single-deletion-correcting codes. *Codes Des.* **2000**, *10*, 273–291.
3. Varshamov, R.; Tenengolts, G. Codes which correct single asymmetric errors. *Autom. I Telemkhanika* **1965**, *161*, 288–292. (In Russian)
4. Tenengolts, G. Nonbinary codes, correcting single deletion or insertion (Corresp.). *IEEE Trans. Inf. Theory* **1984**, *30*, 766–769. [CrossRef]

5. Gabrys, R.; Sala, F. Codes correcting two deletions. *IEEE Trans. Inf. Theory* **2018**, *65*, 965–974. [CrossRef]
6. Sima, J.; Bruck, J. Optimal k-Deletion Correcting Codes. In Proceedings of the 2019 IEEE International Symposium on Information Theory (ISIT), Paris, France, 7–12 July 2019 .
7. Levenshtein, V.I. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **1966**, *10*, 707–710.
8. Levenshtein, V.I. Bounds for deletion/insertion correcting codes. In Proceedings of the IEEE International Symposium on Information Theory, Lausanne, Switzerland, 30 June–5 July 2002; p. 370.
9. Kulkarni, A.A.; Kiyavash, N. Nonasymptotic upper bounds for deletion correcting codes. *IEEE Trans. Inf. Theory* **2013**, *59*, 5115–5130. [CrossRef]
10. Cullina, D.; Kiyavash, N. An improvement to Levenshtein's upper bound on the cardinality of deletion correcting codes. *IEEE Trans. Inf. Theory* **2014**, *60*, 3862–3870. [CrossRef]
11. Lovász, L. On the Shannon capacity of a graph. *IEEE Trans. Inf. Theory* **1979**, *25*, 1–7. [CrossRef]
12. Butenko, S.; Pardalos, P.; Sergienko, I.; Shylo, V.; Stetsyuk, P. Estimating the size of correcting codes using extremal graph problems. In *Optimization*; Springer: New York, NY, USA, 2009; pp. 227–243.
13. Kulkarni, A.A.; Kiyavash, N.; Sreenivas, R. On the Varshamov–Tenengolts construction on binary strings. *Discret. Math.* **2014**, *317*, 79–90. [CrossRef]
14. Sloane, N.J.A. Challenge Problems: Independent Sets in Graphs. Available online: https://oeis.org/A265032/a265032.html (accessed on 3 October 2019).
15. Levenshtein, V. Binary codes capable of correcting spurious insertions and deletion of ones. *Probl. Inf. Transm.* **1965**, *1*, 8–17.
16. Levenshtein, V.I. On perfect codes in deletion and insertion metric. *Discret. Math. Appl.* **1992**, *2*, 241–258. [CrossRef]
17. Ginzburg, B. A number-theoretic function with an application in the theory of coding. *Probl. Kibern.* **1967**, *19*, 249–252.
18. Akiba, T.; Iwata, Y. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.* **2016**, *609*, 211–225. [CrossRef]
19. Hespe, D.; Schulz, C.; Strash, D. Scalable kernelization for maximum independent sets. *J. Exp. Algorithmics (JEA)* **2019**, *24*, 1–16. [CrossRef]
20. San Segundo, P.; Rodríguez-Losada, D.; Jiménez, A. An exact bit-parallel algorithm for the maximum clique problem. *Comput. Oper. Res.* **2011**, *38*, 571–581. [CrossRef]
21. Mitchell, S.; OSullivan, M.; Dunning, I. *PuLP: A Linear Programming Toolkit for Python*; The University of Auckland: Auckland, New Zealand, 2011.
22. Forrest, J.; Ralphs, T.; Vigerske, S.; Kristjansson, B.; Lubin, M.; Santos, H.; Saltzman, M. coin-or/Cbc: Version 2.9.9. Available online: https://dx.doi.org/10.5281/zenodo.1317566 (accessed on 3 October 2019).
23. Bliek1ú, C.; Bonami, P.; Lodi, A. Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In Proceedings of the Twenty-Sixth RAMP Symposium, Tokyo, Japan, 16–17 October 2014; pp. 16–17.
24. Absi, N.; van den Heuvel, W. Worst case analysis of Relax and Fix heuristics for lot-sizing problems. *Eur. J. Oper. Res.* **2019**. [CrossRef]
25. Helber, S.; Sahling, F. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *Int. J. Prod. Econ.* **2010**, *123*, 247–256. [CrossRef]
26. D'Andreagiovanni, F.; Mett, F.; Nardin, A.; Pulaj, J. Integrating LP-guided variable fixing with MIP heuristics in the robust design of hybrid wired-wireless FTTx access networks. *Appl. Soft Comput.* **2017**, *61*, 1074–1087. [CrossRef]
27. Wilbaut, C.; Hanafi, S. New convergent heuristics for 0–1 mixed integer programming. *Eur. J. Oper. Res.* **2009**, *195*, 62–74. [CrossRef]
28. Hanafi, S.; Wilbaut, C. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Ann. Oper. Res.* **2011**, *183*, 125–142. [CrossRef]
29. Blum, C.; Puchinger, J.; Raidl, G.R.; Roli, A. Hybrid metaheuristics in combinatorial optimization: A survey. *Appl. Soft Comput.* **2011**, *11*, 4135–4151. [CrossRef]

30. Sankoff, D.; Kruskal, J. Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. *SIAM Rev.* **1983**, *25*, 201–237.
31. Gally, T.; Pfetsch, M.E.; Ulbrich, S. A framework for solving mixed-integer semidefinite programs. *Optim. Methods Softw.* **2018**, *33*, 594–632. [CrossRef]

© 2019 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).