INFORMS Journal on Computing
Vol. 25, No. 2, Spring 2013, pp. 208–221
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1110.0496

A New Local Search Algorithm for Binary Optimization

Dimitris Bertsimas
Operations Research Center and Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, dbertsim@mit.edu

Dan A. Iancu
Graduate School of Business, Stanford University, Stanford, California 94305, daniancu@stanford.edu

Dmitriy Katz
IBM T. J. Watson Research Center, Yorktown Heights, New York 10598, dkatzrog@us.ibm.com

We develop a new local search algorithm for binary optimization problems, whose complexity and performance are explicitly controlled by a parameter Q, measuring the depth of the local search neighborhood. We show that the algorithm is pseudo-polynomial for general cost vector $\mathbf{c}$, and achieves a $w^2/(2w-1)$ approximation guarantee for set packing problems with exactly w ones in each column of the constraint matrix $\mathbf{A}$, when using $Q = w^2$. Most importantly, we find that the method has practical promise on large, randomly generated instances of both set covering and set packing problems, as it delivers performance that is competitive with leading general-purpose optimization software (CPLEX 11.2).

Key words: programming; integer; algorithms; heuristic
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received September 2008; revised October 2010, June 2011, July 2011, August 2011; accepted November 2011. Published online in *Articles in Advance* April 11, 2012.

1. Introduction

In the last fifty years, there has been considerable progress in our ability to solve large-scale binary optimization problems of the form

$$\begin{aligned} \max \quad & \mathbf{c}'\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \in \{0,1\}^n, \end{aligned} \tag{1}$$

where $\mathbf{A} \in \mathbb{Z}^{m\cdot n}$, $\mathbf{b} \in \mathbb{Z}^m$, and $\mathbf{c} \in \mathbb{Z}^n$ are given data. A testimony to this progress is the fact that major codes (such as CPLEX, EXPRESS, or GUROBI) are now capable of solving such problems that a decade ago were out of reach. In addition to very significant speedups in computing power, the two major ingredients that led to progress on the algorithmic side were (a) the introduction of new cutting plane methods, using a plethora of valid inequalities that improve the bounds on the solution and the ability to prove optimality; and (b) the use of heuristic algorithms. Although it is difficult to make an exact assessment of the merits of each algorithmic development, we believe that new cutting plane methods have had a more significant impact than the use of heuristic methods.

Despite the major progress in the field, we still cannot solve especially large and dense binary problems. In real-world applications, there is a stringent desire to find feasible solutions that improve current practice, without necessarily having a proof of their optimality. Thus, there is a definite need to develop general-purpose methods producing high-quality feasible solutions. There are relatively few methods for general binary integer programming problems (see Aarts and Lenstra 1997 for a review), including the lift-and-project methods (Balas et al. 1993), the pivot-and-complement heuristic (Balas and Martin 1980), the "feasibility pump" (Fischetti et al. 2005), and the "pivot, cut, and dive" heuristic (Eckstein and Nediak 2007). Special-purpose exact or approximate algorithms have been developed for specific binary optimization problems, such as set covering (see Balas and Ho 1980, Beasley 1990, Caprara et al. 1999, Chu and Beasley 1996, Al-Sultan et al. 1996, Aickelin 2002 and the survey Caprara et al. 2000 for details) or set packing (Hurkens and Schrijver 1989, Arkin and Hassin 1998, Jia et al. 2004, Koutis 2005).

In this paper, we develop a new algorithm for binary optimization problems and provide both theoretical

and empirical evidence for its strength. Specifically, our contributions are as follows:

1. The algorithm is genuinely general purpose, in that it does not utilize any special combinatorial structure in Problem (1).

2. The trade-off between complexity and performance of the algorithm is explicitly controlled by a parameter Q, which intuitively measures the depth of the neighborhood in the local search. More precisely, with increasing Q, the algorithm can deliver higher quality solutions, at the expense of higher running time.

3. We show that the running time is bounded by $O(\|\mathbf{c}\|_1^2 \cdot n \cdot \binom{2m}{Q} \cdot \max(m, n))$, i.e., for a fixed Q, the algorithm is pseudo-polynomial for general $\mathbf{c}$ and strongly polynomial when $\mathbf{c} = \mathbf{e}$, a vector of ones.

4. For the unweighted maximum w-set packing problem ($\mathbf{A} \in \{0,1\}^{m \cdot n}$, with w ones on each column, and $\mathbf{b} = \mathbf{c} = \mathbf{e}$), we show that our algorithm achieves a $w^2/(2w-1)$ approximation guarantee, slightly weaker than the best known bound in the literature of $w/2$, due to Hurkens and Schrijver (1989).

5. Most importantly, we numerically investigate the algorithm's performance on randomly generated instances of both set covering and set packing, with very encouraging results.

The structure of the rest of the paper is as follows. In §2 we present the algorithm and give an example (which is further expanded in the paper's online supplement available at http://dx.doi.org/10.1287/ijoc.1110.0496). In §3, we analyze its running time, and in §4, we provide the theoretical guarantee for a class of set packing problems. In §5, we discuss implementation details, and in §6, we provide empirical evidence of the algorithm's strength in several classes of set covering and packing problems.

2. Algorithm

Our algorithm takes as inputs the matrix $\mathbf{A}$, the vectors $\mathbf{b}$ and $\mathbf{c}$, a parameter Q and an initial feasible solution $\mathbf{z}_0$, and constructs a sequence of feasible solutions $\mathbf{z}$ with monotonically increasing objective function values. The parameter Q controls the trade-off between the quality of the final output solution and the computational complexity of the algorithm.

2.1. Notation

For any vector $\mathbf{x} \in \{0,1\}^n$, we define the following:

- $\mathbf{x}_v = \max(\mathbf{A}\mathbf{x} - \mathbf{b}, \mathbf{0}) \in \mathbb{Z}_+^m$: the amount of constraint violation produced by $\mathbf{x}$;
- $\mathbf{x}_u = \max(\mathbf{b} - \mathbf{A}\mathbf{x}, \mathbf{0}) \in \mathbb{Z}_+^m$: the amount of constraint slack created by $\mathbf{x}$;
- $\mathbf{x}_w = \min(\mathbf{x}_u, \mathbf{e}) \in \{0,1\}^m$;
- $trace(\mathbf{x}) = [\mathbf{x}_v; \mathbf{x}_w] \in \mathbb{Z}_+^m \times \{0,1\}^m$.

Furthermore, we introduce the following concepts:

- Two solutions $\mathbf{x}$ and $\mathbf{y}$ are said to be *adjacent* if $\mathbf{e}' = 1$.
- A feasible solution $\mathbf{z}_1$ is said to be *better* than another feasible solution $\mathbf{z}_2$ if $\mathbf{c}'\mathbf{z}_1 > \mathbf{c}'\mathbf{z}_2$.
- Let $\mathbf{z}$ be the best feasible solution available to the algorithm at a particular iteration. A solution $\mathbf{y}$ is called *interesting* if the following three properties hold:

(A1) $\|\mathbf{y}_v\|_\infty \le 1$: no constraint is violated by more than one unit;

(A2) $\|trace(\mathbf{y}) - trace(\mathbf{z})\|_1 \le Q$: the total amount of violation in $\mathbf{y}$ plus the number of different loose constraints (as compared to $\mathbf{z}$) is at most Q;

(A3) $\mathbf{c}'\mathbf{y} > \mathbf{c}'\mathbf{x}$, $\forall \mathbf{x}$ already examined by the algorithm, satisfying $h(trace(\mathbf{x})) = h(trace(\mathbf{y}))$.

Here, h: $\{0,1\}^{2m} \to \mathbb{N}$ is a function mapping traces of interesting solutions into integers. Note that, because h is only applied to *interesting* solutions $\mathbf{y}$, which, by condition (A1), must satisfy $\mathbf{y}_v \in \{0,1\}^m$, we can take h: $\{0,1\}^{2m}$, instead of h: $\mathbb{Z}_+^m \times \{0,1\}^m$. The only restriction we impose on $h(\cdot)$ is that evaluating it should be linear in the size of the input: $O(m)$. Apart from that, it can be injective, in which case only solutions with identical traces will be compared, or it can be a hash function (for an introduction to hash functions, see Cormen et al. 2001). The reason for introducing such a hash function is to accelerate the algorithm, at the potential expense of worsening the performance. We will elaborate more on this trade-off in §5, which is dedicated to implementation details.

Note that because of condition (A3), for every value i in the range of h, the algorithm needs to store the highest objective function value of an interesting solution $\mathbf{x}$ satisfying $h(trace(\mathbf{x})) = i$. We will refer to the location where this value is stored as the *trace box* ($\mathcal{TB}$) corresponding to $\mathbf{x}$ or to $trace(\mathbf{x})$, and will denote it by $\mathcal{TB}[i]$.

- The set of interesting solutions is also referred to as the *solution list* ($\mathcal{SL}$).

2.2. Algorithm Outline

With these definitions, we now give a brief outline of the algorithm, which will also give some insight into the types of data structures that are needed. The key ingredients in the heuristic are interesting solutions. In a typical iteration, the algorithm will pick a candidate $\mathbf{x}$ from the list of interesting solutions ($\mathcal{SL}$), and examine all solutions $\mathbf{y}$ adjacent to it. If these solutions turn out to be interesting themselves, they are stored in the list, and the appropriate trace boxes are changed.

By following this method, occasionally we come across solutions $\mathbf{y}$ that are feasible. If they are also better than the best current feasible solution $\mathbf{z}$, then

$\mathbf{z}$ is replaced, the list and the trace boxes are cleared, and the procedure resumes by examining solutions adjacent to $\mathbf{z}$. A formal statement follows.

Algorithm 1 (local search heuristic)
Input: matrix $\mathbf{A}$; vectors $\mathbf{b}, \mathbf{c}$; feasible solution $\mathbf{z}_0$; scalar parameter $Q > 0$
Output: Feasible solution $\mathbf{z}$ such that $\mathbf{c}'\mathbf{z} \geq \mathbf{c}'\mathbf{z}_0$
OPTIMIZEIP($\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{z}_0, Q$)
(1) $\mathbf{z} := \mathbf{z}_0$; $\mathcal{SL} := \{\mathbf{z}\}$
(2) **while** ($\mathcal{SL} \neq \varnothing$)
(3) get a new solution $\mathbf{x}$ from $\mathcal{SL}$
(4) **foreach** ($\mathbf{y}$ adjacent to $\mathbf{x}$)
(5) **if** ($\mathbf{y}$ is feasible) **and** ($\mathbf{c}'\mathbf{y} > \mathbf{c}'\mathbf{z}$)
(6) $\mathbf{z} \leftarrow \mathbf{y}$
(7) $\mathcal{SL} \leftarrow \{\mathbf{y}\}$; $\mathcal{TB}[i] \leftarrow -\infty, \forall i$
(8) **goto** Step 3
(9) **else if** ($\mathbf{y}$ is interesting)
(10) $\mathcal{TB}[h(trace(\mathbf{y}))] \leftarrow \mathbf{c}'\mathbf{y}$
(11) $\mathcal{SL} \leftarrow \mathcal{SL} \cup \{\mathbf{y}\}$
(12) **return** $\mathbf{z}$.

To understand the steps in the algorithm, consider the following set packing problem:

$$\begin{aligned} \max \ & \{x_1 + x_2 + x_3\} \\ \text{s.t.} \ & x_1 \qquad + x_3 \leq 1 \\ & \qquad x_2 + x_3 \leq 1 \\ & x_1, x_2, x_3 \in \{0, 1\}. \end{aligned} \tag{2}$$

It is easy to see, by inspection, that the optimal solution is $\mathbf{x}_{\text{opt}} \stackrel{\text{def}}{=} [x_1, x_2, x_3] = [1, 1, 0]$. To illustrate the steps that Algorithm 1 would take in finding this solution, we will make the following choice concerning the parameters and implementation:

- We will make the simplest possible run, with a parameter $Q = 1$.
- We will start the algorithm with the initial solution $\mathbf{z}_0 = [0, 0, 0]$.
- Because every trace of an interesting solution $\mathbf{x}$ is a binary vector, $trace(\mathbf{x}) \equiv [t_{2m-1}, t_{2m-2}, \ldots, t_1, t_0] \in \{0, 1\}^{2m}$, we will take the mapping $h(\cdot)$ to be the real value corresponding to this binary string, i.e.,

$$h\colon \{0, 1\}^{2m} \to \mathbb{R},\ h([t_{2m-1}, \ldots, t_1, t_0]) = \sum_{i=0}^{2m-1} t_i \cdot 2^i.$$

- We will assume that the solution list $\mathcal{SL}$ is implemented as a first-in, first-out (FIFO) list, so that solutions are extracted in the same order in which they are inserted.

With these remarks, we now proceed to list the first few steps in the algorithm:

- (Step 1) $\mathbf{z} := [0, 0, 0]$; $\mathcal{SL} := \{[0, 0, 0]\}$.
- (Step 2) $\mathcal{SL} := \{[0, 0, 0]\} \neq \varnothing$.
- (Step 3) $\mathbf{x} \leftarrow [0, 0, 0]$. Adjacent solutions are $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$.
  —(Step 4) $\mathbf{y} = [1, 0, 0]$, $trace(\mathbf{y}) = [0, 0; 0, 1]$.
  ∗ (Step 5) $\mathbf{y}$ feasible, $\mathbf{e}'\mathbf{y} = 1 > \mathbf{e}'\mathbf{z}$.
  ∗ (Steps 6–7) $\mathbf{z} \leftarrow [1, 0, 0]$; $\mathcal{SL} \leftarrow \{[1, 0, 0]\}$; $\mathcal{TB}[i] \leftarrow -\infty, \forall i$.
- (Step 3) $\mathbf{x} \leftarrow [1, 0, 0]$. Adjacent solutions are $[0, 0, 0]$, $[1, 1, 0]$, $[1, 0, 1]$.
  —(Step 4) $\mathbf{y} = [0, 0, 0]$, $trace(\mathbf{y}) \stackrel{\text{def}}{=} [\mathbf{y}_v; \mathbf{y}_w] = [0, 0; 1, 1]$.
  ∗ (Step 5) $\mathbf{y}$ feasible, but $\mathbf{e}'\mathbf{y} = 1 = \mathbf{e}'\mathbf{z}$.
  ∗ (Step 9) $\mathbf{y}$ is found to be interesting, because:
  (A1) true: $\|\mathbf{y}_v\|_\infty = \|[0, 0]\|_\infty \leq 1$.
  (A2) true: $\|trace(\mathbf{y}) - trace(\mathbf{z})\|_1 \leq Q$.
  (A3) true: $\mathbf{e}'\mathbf{y} = 1 > \mathcal{TB}[h(trace(\mathbf{y}))] = \mathcal{TB}[3] = -\infty$.
  ∗ (Steps 10–11) $\mathcal{TB}[3] \leftarrow 1$; $\mathcal{SL} \leftarrow \{[0, 0, 0]\}$.
  —(Step 4) $\mathbf{y} = [1, 1, 0]$, $trace(\mathbf{y}) = [0, 0; 0, 0]$.
  ∗ (Step 5) $\mathbf{y}$ feasible, $\mathbf{e}'\mathbf{y} = 2 > \mathbf{e}'\mathbf{z}\,(= 1)$.
  ∗ (Steps 6–7) $\mathbf{z} \leftarrow [1, 1, 0]$; $\mathcal{SL} \leftarrow \{[1, 1, 0]\}$; $\mathcal{TB}[i] \leftarrow -\infty, \forall i$.

We note that, although the algorithm has found the optimal solution $\mathbf{z} = \mathbf{x}_{\text{opt}} = [1, 1, 0]$, quite a few steps remain, which we have listed, for completeness, in the online supplement. Moreover, the particular choices of implementation in the above example have been made in order to facilitate exposition and are by no means efficient. In §5, we include a detailed discussion of the data structures and hash functions used in our implementation.

3. Running Time

In this section, we bound the running time of the algorithm as follows:

THEOREM 1. *For fixed Q and injective $h(\cdot)$, the running time of Algorithm 1 is bounded above by*

$$O\left(\|\mathbf{c}\|_1^2 \cdot n \cdot \binom{2m}{Q} \cdot \max(m, n)\right). \tag{3}$$

We postpone the proof of Theorem 1 until the end of this section, and first introduce the following lemma:

LEMMA 1. *The total number of solutions $\mathbf{x}$ that can be examined between two successive updates of the current feasible solution $\mathbf{z}$ is $O(\binom{2m}{Q} \cdot \|\mathbf{c}\|_1)$.*

PROOF. First note that whenever the current feasible solution $\mathbf{z}$ is updated, the solution list $\mathcal{SL}$ is emptied, the trace boxes are cleared, and only $\mathbf{z}$ is inserted in $\mathcal{SL}$. Hence for any solution $\mathbf{x} \neq \mathbf{z}$ to be examined, it must first be inserted into $\mathcal{SL}$.

By condition (A3) in the definition of interesting solutions, an *interesting* $\mathbf{x}$ inserted into $\mathcal{SL}$ must

satisfy $\mathbf{c}'\mathbf{x} > \mathcal{TB}[i]$, where $i = h(trace(\mathbf{x}))$. Because $\mathbf{x} \in \{0,1\}^n$, $\mathbf{c}'\mathbf{x} \in \{\sum_{c_i<0} c_i, \ldots, \sum_{c_i>0} c_i\}$. Hence the number of updates for any trace box i is at most $\|\mathbf{c}\|_1 + 1$, implying that at most $\|\mathbf{c}\|_1 + 1$ different solutions $\mathbf{x}$ mapping to i can be inserted into $\mathcal{SL}$.

The number of trace boxes i is upper bounded by the number of distinct traces of interesting solutions. If $\mathbf{x}$ is an interesting solution, then

- condition (A1) $\Rightarrow \|\mathbf{x}_v\|_\infty \le 1 \Rightarrow \mathbf{x}_v \in \{0,1\}^m \Rightarrow trace(\mathbf{x}) \in \{0,1\}^{2m}$;
- condition (A2) $\Rightarrow \|trace(\mathbf{x}) - trace(\mathbf{z})\|_1 = \|\mathbf{x}_v - \mathbf{0}\|_1 + \|\mathbf{x}_w - \mathbf{z}_w\|_1 \le Q$.

The number of binary vectors of length $2m$ satisfying this property is upper bounded by

$$\binom{2m}{Q} + \binom{2m}{Q-1} + \cdots + \binom{2m}{1} + 1. \tag{4}$$

Thus, there are $O(\binom{2m}{Q})$ trace boxes to keep track of. Because for each trace box at most $\|\mathbf{c}\|_1 + 1$ solutions can be inserted in $\mathcal{SL}$, we conclude that the number of solutions that can be examined, which is always less than the number of solutions inserted in the list, is $O(\binom{2m}{Q} \cdot \|\mathbf{c}\|_1)$. □

The following lemma deals with the amount of computation performed when examining an interesting solution $\mathbf{x}$.

Lemma 2. *The number of operations performed for any interesting solution* $\mathbf{x}$ *that is examined between two consecutive updates of the current feasible solution* $\mathbf{z}$ *is* $O(n \cdot \max(m,n))$.

Proof. Without going into the details of the implementation, let us consider what operations are performed when examining an interesting solution $\mathbf{x}$.

(B1) $trace(\mathbf{x})$ is calculated. This implies the following:

- Computing $\mathbf{A}\mathbf{x} - \mathbf{b}$, which requires $O(m \cdot n)$ operations for a dense matrix $\mathbf{A}$.
- Comparing $\mathbf{A}\mathbf{x} - \mathbf{b}$ with $\mathbf{0}$, to check for violated or loose constraints, requiring $O(m)$ computations.

(B2) Computing the objective function for x, requiring $O(n)$ operations.

(B3) Examining all the solutions $\mathbf{y}$ adjacent to $\mathbf{x}$. One such examination entails:

- Computing $trace(\mathbf{y})$ from $trace(\mathbf{x})$. Because $\mathbf{y} = \mathbf{x} \pm \mathbf{e}_i \Rightarrow \mathbf{A}\mathbf{y} - \mathbf{b} = \mathbf{A}\mathbf{x} - \mathbf{b} \pm \mathbf{A}_\mathbf{i}$. Because $\mathbf{A}\mathbf{x} - \mathbf{b}$ is already available, computing $trace(\mathbf{y})$ only requires $O(m)$ operations.
- Computing the trace box for $\mathbf{y}$, $\mathcal{TB}[h(trace(\mathbf{y}))]$. As mentioned earlier, we are requiring that an evaluation of the function $h(\cdot)$ should use $O(m)$ operations, i.e., linear in the size of the argument. Thus $\mathcal{TB}[h(trace(\mathbf{y}))]$ can be computed with $O(m)$ operations.
- Computing the objective function value for $\mathbf{y}$. This is $O(1)$, because $\mathbf{c}'\mathbf{y} = \mathbf{c}'\mathbf{x} \pm \mathbf{c}_i$.
- Comparing $\mathbf{c}'\mathbf{y}$ with $\mathcal{TB}[h(trace(\mathbf{y}))]$. Because the theorem assumes that the current feasible solution $\mathbf{z}$ is not updated, the results of the examination could be that (i) $\mathbf{y}$ is ignored or (ii) $\mathcal{TB}[h(trace(\mathbf{y}))]$ is replaced and $\mathbf{y}$ is added to $\mathcal{SL}$. Overall complexity is at most $O(n)$.

Because the number of solutions $\mathbf{y}$ adjacent to a given $\mathbf{x}$ is n, the overall complexity of step (B3) is $O(n \cdot \max(m,n))$, dominating steps (B1) and (B2). We conclude that the overall complexity associated with examining any interesting solution $\mathbf{x}$ is $O(n \cdot \max(m,n))$. □

With the help of the preceding lemmas, we can now prove Theorem 1.

Proof. From Lemma 1, the number of solutions that have to be examined between two successive updates of $\mathbf{z}$ is $O(\binom{2m}{Q} \cdot \|\mathbf{c}\|_1)$. From Lemma 2, each such examination entails $O(n \cdot \max(m,n))$ operations. Hence the amount of operations that are performed while examining interesting solutions between updates of $\mathbf{z}$ is $O(\binom{2m}{Q} \cdot \|\mathbf{c}\|_1 \cdot n \cdot \max(m,n))$.

Each update of the current feasible solution $\mathbf{z}$ involves copying the new solution ($O(n)$), emptying the solution list, and clearing the trace boxes. The latter operations are linear in the total number of trace boxes, which, from a result in Lemma 1, is $O(\binom{2m}{Q})$. Therefore updating $\mathbf{z}$ entails $O(\max\{n, \binom{2m}{Q}\})$ operations.

Because $\mathbf{z} \in \{0,1\}^n \Rightarrow \mathbf{c}'\mathbf{z} \in \{\sum_{c_i<0} c_i, \ldots, \sum_{c_i>0} c_i\}$. Hence, there can be at most $\|\mathbf{c}\|_1 + 1$ updates of $\mathbf{z}$. Therefore, the total running time of the algorithm is

$$O\left(\|\mathbf{c}\|_1 \left[\binom{2m}{Q} \cdot \|\mathbf{c}\|_1 \cdot n \cdot \max(m,n) + \max\left\{n, \binom{m}{Q}\right\}\right]\right)$$
$$= O\left(\|\mathbf{c}\|_1^2 \cdot n \cdot \binom{2m}{Q} \cdot \max(m,n)\right). \quad \square$$

We make the observation that when $\mathbf{c} = \mathbf{e}$, the above result becomes $O(n^3 \cdot \binom{2m}{Q} \cdot \max(m,n))$, proving that Algorithm 1 is strongly polynomial for a fixed Q.

4. Performance Guarantee for Set Packing Problems

So far, we have put no restrictions on the particular data structures that are used. Although this level of generality was appropriate for the algorithm description, in order to prove a meaningful result about the performance, we have to be more specific about the details.

As such, for the remaining part of this section, we consider a solution list $\mathcal{SL}$ implemented as a FIFO list, and we consider the ideal case of an injective $h(\cdot)$,

namely, when each trace box corresponds to a unique trace of an interesting solution, and, implicitly, only solutions having exactly the same trace are compared.

We focus on the following binary optimization problem, which is an integer programming formulation for the well-known unweighted w-set packing problem:

$$\begin{aligned} \max \quad & \mathbf{e}'\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \le \mathbf{e} \\ & \mathbf{x} \in \{0,1\}^n, \end{aligned} \tag{5}$$

where $\mathbf{A} \in \{0,1\}^{m\cdot n}$ is additionally restricted so that each variable x_i participates in exactly w constraints, i.e.,

$$\mathbf{e}'\mathbf{A} = w\mathbf{e}'. \tag{6}$$

Let $\mathbf{z}^*$ denote an optimal solution to Problem (5) and $Z^* = \mathbf{e}'\mathbf{z}^*$ be its associated objective function value. Then the following theorem holds:

Theorem 2. *If $Q = w^2$ and $w > 1$, Algorithm* 1*, operating with a FIFO list and an injective $h(\cdot)$, finds a feasible solution* $\mathbf{z}$ *for Problem* (5) *with objective function value $Z_H = \mathbf{e}'\mathbf{z}$ satisfying*

$$\frac{Z^*}{Z_H} \le \frac{w^2}{2w-1}. \tag{7}$$

We defer the proof of the theorem to the end of the section, and first introduce a lemma summarizing the properties of a feasible $\mathbf{z}$ not satisfying the requirement of Theorem 2. In what follows, $\mathbf{a}_j'$ will always denote the jth row of the matrix $\mathbf{A}$, $\mathbf{e}_i$ will denote the ith unit vector, and $\mathbf{e}$ will denote the vector with 1 in every component.

Lemma 3. *Let $\mathbf{z}$ be a feasible solution such that $\mathbf{e}'\mathbf{z} < ((2w-1)/w^2)Z^*$ and $\mathbf{e}'\mathbf{z} \ge \mathbf{e}'\mathbf{y}$, for all solutions $\mathbf{y}$ feasible and adjacent to $\mathbf{z}$. Also let*

$$\mathcal{O} = \{i \in \{1,\dots,n\}\colon \mathbf{z}_i^* = 1\}, \quad (\textit{components} = 1 \textit{ in the optimal solution}) \tag{8}$$

$$\mathcal{I} = \{i \in \{1,\dots,n\}\colon \mathbf{z}_i = 1\}, \quad (\textit{components} = 1 \textit{ in the current solution}) \tag{9}$$

$$\mathcal{V}_i = \{l \in \{1,\dots,m\}\colon \mathbf{a}_l'(\mathbf{z}+\mathbf{e}_i) > 1\}, (i \in \mathcal{O}), \quad (\textit{constraints violated by increasing ith component of current solution}) \tag{10}$$

$$\mathcal{V} = \{i \in \mathcal{O}\colon |\mathcal{V}_i| = 1\}, \tag{11}$$

$$R = |\mathcal{V}|;\ \mathcal{V} \equiv \{v_1,\dots,v_R\}. \tag{12}$$

Then the following properties hold:

$$\mathbf{A}(\mathbf{e}_i+\mathbf{e}_j) \le \mathbf{e} \quad \textit{and} \quad \mathcal{V}_i \cap \mathcal{V}_j = \varnothing, \quad \forall i \ne j \in \mathcal{O}, \tag{13}$$

$$R > \frac{Z^*}{w}, \tag{14}$$

$$v_i \in \mathcal{V} \Rightarrow v_i \notin \mathcal{I} \quad \textit{and} \quad \exists p_i \in \mathcal{I}\backslash\mathcal{O} \text{ s.t. } \mathbf{A}(\mathbf{z}+\mathbf{e}_{v_i}-\mathbf{e}_{p_i}) \le \mathbf{e}, \tag{15}$$

$$\exists j \in \{1,\dots,m\} \quad \textit{and} \quad \exists T \le R \text{ s.t. } \mathbf{A}\left(\mathbf{z}+\sum_{i=1}^{T}\mathbf{e}_{v_i}-\sum_{i=1}^{T}\mathbf{e}_{p_i}+\mathbf{e}_j\right) \le \mathbf{e}. \tag{16}$$

Proof. From the definition of $\mathcal{O}$, $\mathbf{z}^* = \sum_{i\in\mathcal{O}}\mathbf{e}_i$. Because $\mathbf{z}^*$ is feasible, $\mathbf{A}\mathbf{z}^* \le \mathbf{e}$. With $\mathbf{A} \in \{0,1\}^{m\cdot n} \Rightarrow \mathbf{A}(\mathbf{e}_i+\mathbf{e}_j) \le \mathbf{e}$, $\forall i \ne j \in \mathcal{O}$. Intuitively, this means that two variables, x_i and x_j, cannot participate in the same constraint.

To prove the second part of (13), assume, for the purposes of a contradiction, that $\exists l \in \mathcal{V}_i \cap \mathcal{V}_j \Rightarrow \mathbf{a}_l'(\mathbf{z}+\mathbf{e}_i) > 1$ and $\mathbf{a}_l'(\mathbf{z}+\mathbf{e}_j) > 1$. Because $\mathbf{z}$ is feasible, $\mathbf{a}_l'\mathbf{z} \le 1 \Rightarrow \mathbf{a}_l'\mathbf{e}_i = \mathbf{a}_l'\mathbf{e}_j = 1$, in contradiction with the result in the previous paragraph.

To prove (14), first note that only constraints $\mathbf{a}_l$ that are tight at $\mathbf{z}$ can belong to $\mathcal{V}_i$:

$$\forall i \in \mathcal{O}, \forall l \in \mathcal{V}_i, \quad \mathbf{a}_l'(\mathbf{z}+\mathbf{e}_i) > 1 \Rightarrow (\text{because } \mathbf{a}_j'\mathbf{e}_i \le 1, \forall j) \Rightarrow \mathbf{a}_l'\mathbf{z} = \mathbf{a}_l'\mathbf{e}_i = 1. \tag{17}$$

Because each variable participates in exactly w constraints, and $\mathbf{e}'\mathbf{z} < ((2w-1)/w^2)Z^*$, the number of constraints that are tight at $\mathbf{z}$ always satisfies

$$(\text{no. of constraints tight at } \mathbf{z}) < w\cdot\frac{2w-1}{w^2}Z^* = \left(2-\frac{1}{w}\right)Z^*. \tag{18}$$

Now consider the sets $\mathcal{V}_i$. Because $\mathbf{e}'\mathbf{z}^* = Z^*$, there are Z^* such sets, one for each $i \in \mathcal{O}$. If $\exists i \in \mathcal{O}$ s.t. $\mathcal{V}_i = \varnothing$, then $\mathbf{z}+\mathbf{e}_i$ would be feasible, with a strictly larger objective function than $\mathbf{z}$, in contradiction with the second assumption concerning $\mathbf{z}$. Therefore $|\mathcal{V}_i| \ge 1$, $\forall i \in \mathcal{O}$, implying

$$\sum_{i=1}^{n}|\mathcal{V}_i| = \sum_{i:|\mathcal{V}_i|=1}|\mathcal{V}_i| + \sum_{i:|\mathcal{V}_i|\ge 2}|\mathcal{V}_i| \ge R + 2(Z^*-R) = 2Z^* - R. \tag{19}$$

We have argued that only constraints that $\mathbf{z}$ satisfies with equality can belong to $\mathcal{V}_i$. Thus, from (18) and (19), we obtain the desired relation (14):

$$2Z^* - R \le \sum_{i=1}^{n}|\mathcal{V}_i| < (\text{because } \mathcal{V}_i \text{ are disjoint sets, by (13)}) < Z^*\left(2-\frac{1}{w}\right) \Leftrightarrow R > \frac{Z^*}{w}.$$

To prove (15), observe that if $v_i \in \mathcal{V}$, then $v_i \in \mathcal{O}$ and $|\mathcal{V}_{v_i}| = 1$. Then (17) implies that $\exists$ unique $l \in \mathcal{V}_{v_i}$ s.t. $\mathbf{a}_l'(\mathbf{z}+\mathbf{e}_{v_i}) > 1$ and $\forall j \ne l$, $\mathbf{a}_j'(\mathbf{z}+\mathbf{e}_{v_i}) \le 1$.

Assume $v_i \in \mathcal{I}$. Then $\mathbf{z} \ge \mathbf{e}_{v_i}$. Because each variable participates in w constraints, $\exists l_1,\dots,l_w$ distinct

constraints s.t. $\mathbf{a}_{l_j}' \mathbf{e}_{v_i} = 1$, which implies $\mathbf{a}_{l_j}'(\mathbf{z}+\mathbf{e}_{v_i}) \geq 2$, $\forall j = 1, \ldots, w$, in contradiction with $|\mathcal{V}_{v_i}| = 1$. Therefore, $v_i \notin \mathcal{I}$.

Consider again the unique l s.t. $\mathbf{a}_l'(\mathbf{z}+\mathbf{e}_{v_i}) > 1$. From (17), $\mathbf{a}_l'\mathbf{z} = 1 \Rightarrow \exists p_i \in \mathcal{I}$ s.t. $\mathbf{a}_l'\mathbf{e}_{p_i} = 1$. Also, because $v_i \notin \mathcal{I}$, $p_i \neq v_i$. Assume $p_i \in \mathcal{O}$; then $\mathbf{a}_l'\mathbf{e}_{p_i} = \mathbf{a}_l'\mathbf{e}_{v_i} = 1$, $p_i, v_i \in \mathcal{O}$, in direct contradiction with (13). Hence $p_i \notin \mathcal{O}$.

Now consider $\hat{\mathbf{z}} = \mathbf{z} + \mathbf{e}_{v_i} - \mathbf{e}_{p_i}$. $\forall j \in \{1, \ldots, m\}$, $j \neq l$, $\mathbf{a}_j'\hat{\mathbf{z}} \leq \mathbf{a}_j'(\mathbf{z}+\mathbf{e}_{v_i}) \leq 1$. Also, $\mathbf{a}_l'\hat{\mathbf{z}} = 1+1-1 = 1$. Therefore, $\hat{\mathbf{z}}$ is feasible, concluding the last part of (15).

Before establishing the proof of (16), first note that result (15) can be extended by induction if $p_i \neq p_j$ when $v_i \neq v_j$. Namely, $\forall T \leq R$, the following solution $\hat{\mathbf{z}}$ will be feasible:

$$\hat{\mathbf{z}} = \mathbf{z} + \sum_{i=1}^{T} \mathbf{e}_{v_i} - \sum_{i=1}^{T} \mathbf{e}_{p_i}. \tag{20}$$

If, for some $v_i \neq v_j$, we have $p_i = p_j$, then an even stronger statement holds: $\hat{\mathbf{z}} = \mathbf{z} - \mathbf{e}_{p_i} + \mathbf{e}_{v_i} + \mathbf{e}_{v_j}$ will be feasible (because subtracting $\mathbf{e}_{p_i}$ will introduce slack in both constraints, instead of just one), and therefore $T = 1$ and $j = v_j$ satisfy (16).

So for the remaining proof of (16), we can restrict attention to the most general case of $v_i \neq v_j \Rightarrow p_i \neq p_j$. Let us define the following sets:

$$\hat{\mathcal{I}} = \{i \in \mathcal{O}\colon \hat{\mathbf{z}}_i = 1\}, \tag{21}$$

$$\beta = \{l \in \{1, \ldots, m\}\colon \exists i \in \hat{\mathcal{I}} \text{ s.t. } \mathbf{a}_l'\mathbf{e}_i = 1\}, \tag{22}$$

$$\bar{\beta} = \{1, \ldots, m\} \backslash \beta. \tag{23}$$

The set of all variables $\hat{\mathcal{I}}$ are 1 in both $\hat{\mathbf{z}}$ and the optimal solution, $\mathbf{z}^*$. From the construction of $\hat{\mathbf{z}}$, it can be seen that $\hat{\mathcal{I}} = \mathcal{V} \cup (\mathcal{I} \cap \mathcal{O})$. From (17), $\forall v_i \in \mathcal{V} \Rightarrow v_i \notin \mathcal{I} \Rightarrow \mathcal{V} \cap \mathcal{I} = \varnothing \Rightarrow |\hat{\mathcal{I}}| = |\mathcal{V}| + |\mathcal{I} \cap \mathcal{O}|$. Letting $n_0 = |\mathcal{I} \cap \mathcal{O}|$, we have $|\hat{\mathcal{I}}| = R + n_0$.

The set of all constraints β is where variables from $\hat{\mathcal{I}}$ participate. Because $\forall i \neq j \in \hat{\mathcal{I}} \Rightarrow i, j \in \mathcal{O}$, then, from (13), they cannot participate in the same constraint, so $|\beta| = (R + n_0)w$.

The set of all other constraints $\bar{\beta}$, $|\bar{\beta}| = m - w(R+n_0)$.

From (20), with $T = R$, we obtain that $\hat{\mathbf{z}} = \mathbf{z} + \sum_{i=1}^{R} \mathbf{e}_{v_i} - \sum_{i=1}^{R} \mathbf{e}_{p_i}$ is feasible. Because $\mathbf{e}'\hat{\mathbf{z}} = \mathbf{e}'\mathbf{z} < (Z^*/w)(2-1/w)$, then, by an argument similar to (18), the number of tight constraints in $\hat{\mathbf{z}}$ is $< Z^*(2-1/w)$. Furthermore, because $\hat{\mathbf{z}}_i = 1$, $\forall i \in \hat{\mathcal{I}}$, all the β constraints are tight, so the number of tight $\bar{\beta}$ constraints is $< Z^*(2-1/w) - (R+n_0)w$. From (14),

$$\begin{aligned} R > Z^*/w \;\Rightarrow\; & Z^*\left(2 - \frac{1}{w}\right) - (R+n_0)\cdot w \\ & \leq Z^*\left(2 - \frac{1}{w}\right) - R \cdot w - n_0 \\ & < Z^*\left(2 - \frac{1}{w}\right) - \frac{Z^*}{w}(w-1) - R - n_0 \\ & = Z^* - R - n_0. \end{aligned} \tag{24}$$

Now consider all the variables in $\mathcal{O} \backslash \hat{\mathcal{I}}$. For any such variable j, $j \notin \hat{\mathcal{I}} \Rightarrow \hat{\mathbf{z}}_j = 0$ and j only participates in $\bar{\beta}$ constraints. Also, $\forall i \neq j \in \mathcal{O} \backslash \hat{\mathcal{I}}$, from (13), j and i cannot participate in the same constraint. But from (24), there are $< Z^* - R - n_0$ tight constraints involving variables j, and there are $|\mathcal{O}| - |\hat{\mathcal{I}}| = Z^* - R - n_0$ such j. Therefore $\exists j$ s.t. $\hat{\mathbf{z}} + \mathbf{e}_j$ is feasible, proving (16). □

The main result of the preceding lemma is (16), which indicates that for any solution $\mathbf{z}$ not satisfying the requirements of Theorem 2, a better feasible solution $\hat{\mathbf{z}} = \mathbf{z} + \sum_{i=1}^{R} \mathbf{e}_{v_i} - \sum_{i=1}^{R} \mathbf{e}_{p_i} + \mathbf{e}_j$ can be constructed, by (1) subtracting all the relevant $\mathbf{e}_{p_i}$, (2) adding $\mathbf{e}_j$, and (3) adding all the corresponding $\mathbf{e}_{v_i}$.

However, it is not immediately clear that our algorithm would proceed according to these steps. For instance, perhaps a solution $\mathbf{z} - \sum_{i=1}^{R} \mathbf{e}_{p_i}$ is never examined. As such, we need one more result concerning the reachability of $\hat{\mathbf{z}}$.

We introduce the concept of a *generation*, defined by the following recursion:

- Let the best feasible solution $\mathbf{z}$ always have generation 0.
- For any solution $\mathbf{y}$ inserted in the list at Step 11 of Algorithm 1, define its generation to be 1+the generation of the solution $\mathbf{x}$ from Step 4, to which $\mathbf{y}$ is adjacent.

Observe that the definition is consistent: the generation counting is always reset when the current feasible solution $\mathbf{z}$ is updated in Step 6, because the solution list is cleared and $\mathbf{z}$, whose generation is set to zero, is the only solution added to the list. From that point onward, for any solution $\mathbf{x}$ extracted and examined in Steps 3 and 4, the generation t will simply represent the number of variables that the algorithm has changed starting at $\mathbf{z}$ in order to reach $\mathbf{x}$. Note that this is not the same as the distance between $\mathbf{z}$ and $\mathbf{x}$. For instance, $\mathbf{x} = \mathbf{z} + \mathbf{e}_i - \mathbf{e}_i$ will actually be identical to $\mathbf{z}$, but it will have generation two.

An immediate consequence of this assignment is that all the solutions $\mathbf{x}$ will be inserted into (and hence extracted from) the FIFO list in an increasing order of generations.

With variables $\mathbf{z}$, R and indices p_i, v_i and j having the same significance as that from Lemma 3, we establish the following result:

Lemma 4. *If $Q \geq T \cdot w$ and $T \leq R$, a feasible solution of generation T with the same trace and objective function value as $\mathbf{z} - \sum_{i=1}^{T} \mathbf{e}_{p_i}$ will be in the FIFO list.*

Proof. First note that, as a consequence of (15), $\forall t \leq T$, $\mathbf{z} \geq \sum_{i=1}^{t} \mathbf{e}_{p_i}$, which makes the subtraction operations well defined. Furthermore, any such solution is feasible (because $\mathbf{z}$ itself is feasible), which also implies that any solution with the same trace as $\mathbf{z} \geq \sum_{i=1}^{t} \mathbf{e}_{p_i}$ must also be feasible.

The first step of the induction is trivial: generation zero has $\mathbf{z}$ in the list. Assume that the property holds for solutions of the tth generation, $t < T$, and call such a solution $\mathbf{z}^{(t)}$. Note that $\mathbf{z}^{(t)}$ is not necessarily equal to $\mathbf{z} - \sum_{i=1}^{t} \mathbf{e}_{p_i}$. It only has the same trace and objective function value.

We claim that $\mathbf{z}^{(t)}_{p_{t+1}} = 1$. Assume by contradiction that $\mathbf{z}^{(t)}_{p_{t+1}} = 0$. With any subtraction of $\mathbf{e}_{p_i}$, exactly w constraints become loose, and hence $\|trace(\mathbf{x}) - trace(\mathbf{z})\|_1$ increases by w. Thus the trace distance between $\mathbf{z}^{(t)}$ and $\mathbf{z}$ is exactly tw. If $\mathbf{z}^{(t)}_{p_{t+1}} = 0$, and the trace is the same as that of $\mathbf{z} - \sum_{i=1}^{t} \mathbf{e}_{p_i}$, then in some earlier generation, the variable at p_{t+1} was changed from one to zero. Also, to maintain the same trace, it must have been the case that one other variable was changed from zero to one in each of the w constraints in which p_{t+1} participates. But this would cause a delay of at least two generations as compared to $\mathbf{z}^{(t)}$, meaning that such a solution could not have been already examined. It is the property of the FIFO list that imposes that solutions will be examined in a strictly increasing order of generations. Hence it must be that $\mathbf{z}^{(t)}_{p_{t+1}} = 1$.

But then, in the tth generation, a solution with the same trace and objective function as $\mathbf{z}^{(t)} - \mathbf{e}_{p_{t+1}}$ will be examined. Because $Q \geq T \cdot w \geq (t+1)w$, and this solution is feasible, it will immediately satisfy conditions (A1) and (A2) characterizing interesting solutions. For condition (A3), there are two cases:

- If the objective function value for this solution is larger than that found in its corresponding trace box, the solution will be added to the list, with generation $t+1$ assigned to it, and the induction proof is complete.
- Otherwise, because all the trace boxes are set to $-\infty$ when the list is cleared, it must be that some other solution $\tilde{\mathbf{z}}$, mapping to the same trace box, was already added to the list in some earlier step. Because $h(\cdot)$ is injective, it must be that $trace(\tilde{\mathbf{z}}) = trace(\mathbf{z}^{(t)} - \mathbf{e}_{p_{t+1}})$. But, as argued in the preceding paragraph, this would imply that the distance between $trace(\mathbf{z})$ and $trace(\tilde{\mathbf{z}})$ was exactly $(t+1)w$, meaning that at least $t+1$ variables were changed starting from $\mathbf{z}$ in order to reach $\tilde{\mathbf{z}}$. But then $\tilde{\mathbf{z}}$ must have generation $t+1$, completing our inductive proof. □

With the preceding lemmas, we are now ready to prove the result of Theorem 2.

Proof. Assume the heuristic is run with some initial feasible solution $\mathbf{z} = \mathbf{z}_0$ satisfying $\mathbf{e}'\mathbf{z} < ((2w-1)/w^2)Z^*$. If there are solutions $\mathbf{y}$ adjacent to $\mathbf{z}$ that satisfy the condition at Step 5 in the algorithm (namely, they are feasible and have better objective function value than $\mathbf{z}$), then Steps 6–7 will clear the solution list and replace $\mathbf{z}$ with $\mathbf{y}$. If repeating this process results in a feasible solution $\mathbf{z}$ satisfying Equation (7), then there is nothing to prove. So, without loss of generality, let us assume that we reach a feasible solution $\mathbf{z}$ for which no adjacent $\mathbf{y}$ satisfies the condition at Step 5. Then, from Lemma 3, a feasible solution $\hat{\mathbf{z}} = \mathbf{z} + \sum_{i=1}^{R} \mathbf{e}_{v_i} - \sum_{i=1}^{R} \mathbf{e}_{p_i} + \mathbf{e}_j$ exists.

By Lemma 4, after t generations, a solution $\mathbf{z}^{(t)}$ with the same trace and objective function value as $\mathbf{z} - \sum_{i=1}^{t} \mathbf{e}_{p_i}$ will be in the FIFO list. The number t of such generations that need to be considered is given by the first time when $\mathbf{e}_j$ can be added. Because e_j participates in w constraints, it will collide with at most w of the p_i, which must first be subtracted. Therefore, we require $t \geq w$, which, by Lemma 4, implies that $Q \geq w^2$, justifying the condition in the statement of the theorem.

Once all the p_i's are subtracted, in generation w, a feasible solution with the same trace and objective function as $\mathbf{z} - \sum_{i=1}^{w} \mathbf{e}_{p_i} + \mathbf{e}_j$ will be considered by the algorithm. By the same inductive argument as in the proof of Lemma 4, it can be seen that for all future generations $w+1+t$, a feasible solution with the same trace and objective function value as $\mathbf{z} - \sum_{i=1}^{w} \mathbf{e}_{p_i} + \mathbf{e}_j + \sum_{i=1}^{t} \mathbf{e}_{v_i}$ will be in the FIFO list. After $2w+1$ generations, a feasible solution $\mathbf{z}^{(2w+1)} = \mathbf{z} - \sum_{i=1}^{w} \mathbf{e}_{p_i} + \mathbf{e}_j + \sum_{i=1}^{w} \mathbf{e}_{v_i}$, with objective function $\mathbf{e}'\mathbf{z}^{(2w+1)} = \mathbf{e}'\mathbf{z} + 1$, will be examined. In Step 6, the current feasible solution $\mathbf{z}$ will be replaced with $\mathbf{z}^{(2w+1)}$, and the solution list and trace boxes will be cleared.

Repeating this argument inductively for the new $\mathbf{z}$, we see that the end solution has to obey $Z^*/Z_H \leq w^2/(2w-1)$, proving Theorem 2. □

Two observations are worth making at this point. First, we note that our result is only slightly weaker than the best known bound in the literature for w-set packing, namely, $w/2 + \epsilon$, found in Hurkens and Schrijver (1989). Although this is certainly encouraging, we also remark that, in order to achieve this bound, our algorithm requires $Q \geq w^2$, which would give rise to an $O(m^{w^2})$ term in the running time of Theorem 1. Hence, for large values of w, one should carefully trade off between computational requirements and the desired quality of the objective function value.

5. Implementation

In this section, we present several details specific to our implementation of the algorithm. Although these provide a guideline for several "good" choices of data structures and parameter values, they should by no means be regarded as exhaustive or optimal. Our main reason for including them here is to provide a complete framework for the computational results in §6.

5.1. Problem Representation

To accommodate large data sets, we have opted to implement the constraint matrix $\mathbf{A}$ as a sparse matrix (n sparse vectors, one for each column). The vectors

$\mathbf{b} \in \mathbb{Z}^m$ and $\mathbf{c} \in \mathbb{Z}^n$ were represented as dense vectors (arrays of integer values).

For the solutions $\mathbf{x}$, we have taken two different approaches. In problems where $\mathbf{x} \in \mathbb{Z}^n$, solutions were represented as n-dimensional dense vectors. For problems with $\mathbf{x} \in \{0,1\}^n$, every solution was represented as a bit array (also known as *bit field* or *bitmap*). This compact representation significantly reduced the memory requirements, which turned out to be essential for achieving better performance.

Because the algorithm usually operated with *interesting* solutions, which, by conditions (A1) and (A2) from §2.1, have few nonzero entries, we decided to store the traces of solutions as sparse arrays.

5.2. Algorithm-Characteristic Data Structures

As hinted to in earlier sections, the major choices in terms of implementation were the solution list $\mathcal{SL}$, with the associated trace boxes, and the function $h(\cdot)$.

Note that if no other restrictions are imposed, an implementation using a FIFO solution list, with a large Q, could create structures of very large size, because interesting solutions could be added for a very long time until a feasible solution of better objective function value is found, and the list is cleared. To fix this situation, we have decided to store as many solutions as there are trace boxes. After all, once a solution is deemed *interesting*, the previous solution mapping to the same trace box is no longer interesting, and hence could simply be ignored.

This brings us to the issue of the number of trace boxes. The ideal case of an injective $h(\cdot)$, which implies having one trace box for each possible trace of an interesting solution, would require $O(\binom{2m}{Q})$ boxes, by Equation (4). Because for every trace box, we would also like to store the associated interesting solution, this would imply a memory commitment of $O(n \cdot \binom{2m}{Q})$, which for large m, n could cause problems even in modern systems.

As suggested in §2.1, one way to overcome these difficulties is to relax the requirement of having $h(\cdot)$ injective. Instead, we would consider a function $h\colon U \to V$, where $U \subset \{0,1\}^{2m}$ is the set of traces of interesting solutions and $V = \{1, 2, \ldots, N_{TB}\}$ is the set of indices of trace boxes. The parameter N_{TB} represents the total number of trace boxes that can be considered, which is also the total size of the allowed solution list $\mathcal{SL}$. As such, it provides a direct connection with the total amount of memory committed to the algorithm, and can be adjusted depending on the available resources.

The advantage of this approach is that we are now free to choose N_{TB} and $h(\cdot)$. The main pitfall is that for most practical problems, $N_{TB} << |U|$, and hence multiple interesting solutions with different traces will map to the same trace box, causing some of them to be ignored in the search. If the number of such collisions is high, then the algorithm might ignore many good directions of improvement, resulting in poor performance. To minimize this undesirable effect, we take the following twofold approach:

1. We choose $h(\cdot)$ as a *hash function*, namely a mapping from a large universe of values (U) to a much smaller set (V), with as few collisions as possible.

2. Instead of having a single hash function $h(\cdot)$, i.e., allowing each trace of an interesting solution to map to a single trace box, we consider a family of hash functions $h^i(\cdot)$, $i \in \{1, 2, \ldots, N_H\}$. The parameter N_H, representing the number of distinct trace boxes into which an interesting trace gets mapped, is a fixed, small number that becomes another choice in the design.

With the addition of multiple hash functions $h^i(\cdot)$, the original definition of an *interesting solution* from §2.1 has to be modified slightly. Although the first two conditions remain the same, a solution $\mathbf{y}$ is now found interesting if $\mathbf{c}'\mathbf{y} > \mathbf{c}'\mathbf{x}$ for all $\mathbf{x}$ already examined such that $h^i(trace(\mathbf{x})) = h^i(trace(\mathbf{y}))$ for some $i \in \{1, \ldots, N_H\}$. In other words, in Step 9 of Algorithm 1, $\mathbf{y}$ is interesting if its objective function value $\mathbf{c}'\mathbf{y}$ is larger than at least one of the values stored in the trace boxes $h^1(trace(\mathbf{y}))$, $h^2(trace(\mathbf{y}))$, $\ldots$, $h^{N_H}(trace(\mathbf{y}))$. If that is the case, in Step 10, the value $\mathbf{c}'\mathbf{y}$ is stored in all the trace boxes satisfying this property, and the solution $\mathbf{y}$ is written in the corresponding locations in the solution list at Step 11.

The downside for using this approach is that the theoretical result presented in prior sections change for the worse. Namely, for a general cost vector $\mathbf{c} \in \mathbb{Z}^n$, with the number of trace boxes fixed to N_{TB} and the number of hash functions fixed to N_H, the running time from §3 becomes

$$O(\|\mathbf{c}\|_1^2 \cdot N_{TB} \cdot n \cdot \max(m, n \cdot N_H)), \tag{25}$$

and the performance guarantee from §4 is lost. However, as we will see in §6, this approach is advantageous from a computational perspective, and delivers very good results in practice.

5.3. Hash Functions

To complete the description of the implementation, in this subsection we present our particular choice of functions $h^i(\cdot)$. Although the literature on hash functions is abundant and many good choices are available (see Cormen et al. 2001 for an introduction and Bakhtiari et al. 1995 for a survey article), we have settled for a less sophisticated version, which we describe in the next paragraphs.

In the first step, for each hash function h^i, $i \in \{1, 2, \ldots, N_H\}$, a set of m positive integer values was generated. These values were chosen uniformly at random, and only once, at the very beginning of the

algorithm. Let the ith set of such values be $\Phi^i = \{\phi_1^i, \phi_2^i, \ldots, \phi_m^i\}$.

Given the total (fixed) number N_{TB} of traces boxes, we distinguish the following two regions of equal size:

1. The first region, henceforth referred to as the "y_v region," corresponds to interesting solutions $\mathbf{y}$ with $\mathbf{y}_v \neq 0$ (i.e., violating certain constraints). This region is further split into subregions, depending on the number of violated constraints:

• The first subregion, of size m, corresponds to solutions $\mathbf{y}$ with exactly one violated constraint ($\|\mathbf{y}_v\|_1 = 1$). Because there are m total constraints, the mapping into this region is trivial: a solution that violates only constraint i will be mapped to the ith box of this region.

• The remaining $(N_{TB}/2 - m)$ boxes from the y_v region are split evenly among exactly $Q-1$ subregions. Any interesting solution $\mathbf{y}$, with violated constraints $j_1, j_2, \ldots, j_p$ $(2 \leq p \leq Q)$, would be mapped only to the pth such subregion, and would have N_H boxes corresponding to it, one for each hash function. The ith hash function would compute the corresponding trace box according to the following formula:

$$h^i[trace(\mathbf{y})] = \left(\sum_{k=1}^{p} \phi_{j_k}^i + \prod_{k=1}^{p} \phi_{j_k}^i\right) \bmod \left(\frac{N_{TB}/2 - m}{Q-1}\right), \quad i \in \{1, \ldots, N_H\}, \tag{26}$$

where $(a \bmod b)$ denotes the remainder obtained when dividing the integer a by the integer b. The above formula has a simple interpretation: the first term is a combination of the set Φ^i of random values, based on the indices $j_1, \ldots, j_p$ of the violated constraints. The *mod* operation ensures that the resulting index is in a range suitable for the pth subregion. The intuition behind why the formula works and results in few collisions is more complicated, and is beyond the scope of the current paper (we refer the interested reader to Bakhtiari et al. 1995 for a more comprehensive treatment).

2. The second region, also of size $N_{TB}/2$, corresponds to interesting solutions with no violated constraints ($\mathbf{y}_v = \mathbf{0}$), but with loose constraints ($\mathbf{y}_w \neq \mathbf{0}$). Similar to the previous discussion, this region is called the "y_w region," and is further divided into subregions:

• The first subregion has size m, and corresponds to solutions with exactly one loose constraint. The mapping here is analogous to that from the y_v case.

• The remaining $N_{TB}/2 - m$ boxes are divided evenly among the $Q-1$ subregions corresponding to solutions with more than one loose constraint. However, unlike the situation with y_v, it is no longer desirable to map solutions with p loose constraints exclusively in the pth subregion. Instead, these solutions should also be compared with solutions having fewer than p loose constraints. The intuition is that if a solution having more loose constraints also has higher objective function value, then it would be desirable to have it considered by the algorithm. To accommodate for this new provision, for each solution with loose constraints $j_1, \ldots, j_p$ $(p \geq 2)$, we choose several subsets of 1, 2, or r constraints ($r \leq p$ could be either a function of p or chosen in some deterministic way). The numbers of such subsets, henceforth referred to as N_1, N_2, and N_r, respectively, are fixed and become parameters of the algorithm. Furthermore, the choice of the subsets themselves is done in a deterministic fashion, so that for any particular trace of an interesting solution $\mathbf{y}$, the same subsets are always chosen. Once such a subset of indices $j_1, \ldots, j_r$ is fixed, the trace index is computed with the help of one of the hash functions defined before—for instance, we could use the first hash function:

$$h^1[trace(\mathbf{y})] = \left(\sum_{k=1}^{r} \phi_{j_k}^1 + \prod_{k=1}^{r} \phi_{j_k}^1\right) \bmod \left(\frac{N_{TB}/2 - m}{Q-1}\right). \tag{27}$$

Note that because we are already considering multiple sets of indices, the same solution is automatically mapped into multiple boxes in the y_w region, so there is no need to compute the results from multiple hash functions, as was done for $\mathbf{y}_v$.

We conclude this section by making two relevant observations. First, note that because the "random" values Φ^i do not change during the run of the algorithm, the hash functions $h^i(\cdot)$ are deterministic, in the sense that the same trace of a particular solution $\mathbf{y}$ is always mapped to the same trace boxes, regardless of the time at which it is considered by the algorithm. Therefore, the set of rules specified above uniquely determines the way in which each interesting solution is mapped into the trace boxes (and, implicitly, in the solution list).

Second, observe that the number of trace boxes N_{TB} (or, equivalently, the total amount of memory committed to the solution list) and the parameter Q should, in general, not be chosen independently. The reason is that for a fixed N_{TB}, the size of each subregion in both the y_v and the y_w regions is inversely proportional with Q. Therefore, if we would like the parameter Q to be a good indicator of the performance of the algorithm (i.e., larger Q resulting in improved objective function value), then we should increase N_{TB} accordingly, so that the ratio N_{TB}/Q remains roughly constant.

5.4. Extracting a New Solution from the List

The last relevant detail of the implementation is the way in which interesting solutions are extracted

from the solution list $\mathcal{SL}$, at Step 3 of the algorithm. Although any procedure that extracts solutions repeatedly would eventually explore all interesting solutions, particular choices for the order of extraction could speed up the algorithm considerably. For example, it is desirable to first examine solutions $\mathbf{y}$ adjacent to an interesting solution $\mathbf{x}$ that has few violated constraints, because such directions are more likely to result in feasible solutions.

To this end, we have included in the implementation a simple scheme based on a *priority queue*. The main idea behind this data type is that each element inserted in the queue also has an associated value, which determines its priority relative to the other elements in the queue. Whenever an extraction occurs, the first element to leave the queue is the one with the highest priority among all the members of the queue. For a comprehensive treatment and other references, we refer the interested reader to (Cormen et al. 2001).

To implement this concept in our setting, whenever a solution $\mathbf{y}$ was determined as interesting at Step 9, a priority value $pv(\mathbf{y})$ was computed based on $\mathbf{y}$'s objective function value and the number of constraints it violated (we used a very simple, additive scheme). When $\mathbf{y}$ was written in the solution list at Step 11, the index of its corresponding trace box was introduced in the priority queue, with a priority of $pv(\mathbf{y})$. By following this rule, the solution $\mathbf{x}$ extracted at Step 3 always had the largest priority among all solutions in the list.

The downside for using a priority queue is that we need to store an additional $O(N_{TB})$ values, and the complexity for inserting and/or extracting from the priority queue becomes $O(\log N_{TB})$, hence raising the overall complexity of the scheme. However, despite this seemingly higher computational load, the actual (physical) running time is usually decreased, because the heuristic spends less time searching in "infertile" directions.

5.5. Running the Algorithm

We conclude this section by first summarizing the parameters that the user is free to choose in our implementation of the heuristic:

- Q—the parameter determining what constitutes an interesting solution.
- N_{TB}—the number of trace boxes, also equal to the size of the solution list. Because specifying a particular N_{TB} is equivalent to fixing a certain memory commitment (*MEM*) for the solution list, we have decided to use the latter for convenience.
- N_H—the number of hash functions, influencing how many boxes correspond to each interesting solution.
- N_1, N_2, and N_r—the number of subsets of 1, 2, or r loose constraints, respectively, which should be considered when computing the indices of the trace boxes.

To simplify the benchmarking of the algorithm, we decided to fix some of the adjustable parameters to a choice that consistently delivered good results in our experiments:

$$N_H = 2; \quad N_1 = 2; \quad N_2 = 2; \quad N_r = 5.$$

With respect to the two remaining parameters, Q and *MEM*, we found that the most natural way to run the heuristic procedure is in stages, by gradually increasing the values of both Q and *MEM*. The reason is that cold starting the procedure directly with large values of Q and *MEM* would result in an unnecessarily large computational time spent in clearing the (large) solution list $\mathcal{SL}$, which is done whenever the current feasible solution is updated. Thus, to improve the physical running time, one should always first run the heuristic with smaller values of Q and *MEM*, which would (quickly) deliver better feasible solutions, that could in turn be used to warm start the heuristic with larger Q and *MEM*.

6. Computational Results

We have tested our implementation of the algorithm on several classes of problems, and have compared the results with the output from IBM ILOG CPLEX 11.2. Although the latter is a general-purpose integer optimization solver, and hence a comparison with a (heuristic) algorithm geared toward pure-binary problems might not be deemed entirely fair, we maintain that it is nonetheless very meaningful, particularly given CPLEX's strength and ubiquity.

All our tests were run on the Massachusetts Institute of Technology Operations Research Center computational machine, which is a Dual Core Intel® Xeon® 5050 Processor (3.00 GHz, 4 MB Cache, 667 MHz FSB), with 8 GB of RAM (667 MHz), running Ubuntu Linux.

Consistent with our remarks in the end of §5.5, we used the values $N_H = 2$, $N_1 = 2$, $N_2 = 2$, $N_r = 5$, and the following sequence of runs of the heuristic in all the test cases:

$$\begin{aligned}&(1)\ Q=4, MEM=10\text{ MB} \Rightarrow (2)\ Q=4, MEM=50\text{ MB}\\ &\Rightarrow (3)\ Q=6, MEM=100\text{ MB} \Rightarrow (4)\ Q=6,\\ &MEM=250\text{ MB} \Rightarrow (5)\ Q=10,\\ &MEM=1\text{ GB} \Rightarrow (6)\ Q=10, MEM=2\text{ GB} \Rightarrow\\ &(7)\ \begin{cases} Q=15, MEM=6\text{ GB}\\ Q=20, MEM=6\text{ GB}.\end{cases}\end{aligned}$$

In step (7), the brace indicates that the two independent runs were both started with the same initial feasible solution, given by the output from the run in

step (6). These two runs were still performed sequentially (i.e., *nonconcurrently*), so that the total completion time of the heuristic was given by the sum of the completion times of stages (1) to (6) and the two runs in stage (7).

6.1. Set Covering

The first type of problem that we considered was set covering, i.e.,

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}'\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{e} \\ & \mathbf{x} \in \{0,1\}^n, \end{aligned} \tag{28}$$

where $\mathbf{A} \in \{0,1\}^{m \cdot n}$. To have sufficiently large data sets, we wrote a script that generated different instances of the problem. The script took as arguments the number of constraints (m), the number of variables (n), and the number of nonzero entries (w) in each column of $\mathbf{A}$. In addition, there were two parameters specifying lower and upper bounds on the entries in the cost vector $\mathbf{c}$, for the weighted version of the problem.

In the first class of tests, we considered $m = 1{,}000$, $n = 2{,}500$, and took $w = 3$, $w = 5$, or w generated uniformly at random, in $\{3, \ldots, 7\}$. Table 1 records the results of the simulation for the unweighted case, $\mathbf{c} = \mathbf{e}$, and Table 2 contains the results for the weighted case, where the weights c_i were also generated uniformly at random, with values in $[400, 500]$.

In the second category of tests, summarized in Table 3, we considered a larger problem size, $m = 4{,}000$ and $n = 10{,}000$, with w generated uniformly at random in $\{3, \ldots, 7\}$. We note that, for all the set cover instances, CPLEX was run with its default settings (in particular, the *MIP emphasis* parameter was zero).

Based on the examples presented here and several other runs we have performed, our assessment is that Algorithm 1 tends to outperform CPLEX after approximately 20 hours, and in many cases even earlier, provided that both methods are run with the same amount of memory (6 GB). For shorter running times (one-two hours), CPLEX tends to have an edge, although not in all cases.

6.2. Set Packing

The second problem we considered was set packing, i.e.,

$$\begin{aligned} \max \quad & \mathbf{c}'\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{e} \\ & \mathbf{x} \in \{0,1\}^n, \end{aligned} \tag{29}$$

with $\mathbf{A} \in \{0,1\}^{m \cdot n}$. In this case, we also used a script to generate the test cases. Just as with set covering, we took $m = 1{,}000$, $n = 2{,}500$ and considered cases with $w = 3$, $w = 5$, or w generated independently and uniformly at random between three and seven. The results for the unweighted version of the problem, are reported in Table 4, and the results for weighted problems, with all weights generated independently and uniformly at random, with values between 400 and 500, are found in Table 5.

For all the set packing instances, we decided to compare the performance of Algorithm 1 with two different runs for CPLEX, recorded in columns CPX(0) and CPX(4) of the tables, in which the *MIP emphasis* parameter was either set to its default value of zero or to a value of four, respectively. The main difference is that a setting of zero forces the CPLEX algorithms to "balance" between finding rapid proofs of optimality and high-quality feasible solutions, whereas a setting of four (dubbed "hidden feasibility") forces the solver to work hard(er) toward recovering high-quality feasible solutions that would otherwise be difficult to find (for an exact explanation of the *MIP emphasis* parameter, the interested reader is referred to the CPLEX 11.2 documentation, available from IBM

Table 1 Results for Unweighted Set Covering

	5 hr		10 hr		20 hr	
w	ALG	CPLEX	ALG	CPLEX	ALG	CPLEX
3	346	**344**	344	344	344	**343**
3	346	**343**	344	**343**	344	**342**
3	346	**345**	**344**	345	344	344
3	**343**	345	**343**	345	**343**	344
3	**344**	346	343	343	343	**342**
3	**344**	345	**344**	345	344	**343**
3	343	343	343	343	343	**342**
3	**344**	345	**342**	345	**342**	343
3	**343**	345	**343**	345	**343**	344
3	**344**	346	**342**	346	**342**	344
5	**233**	242	**231**	241	**229**	236
5	**231**	243	**229**	243	**229**	236
5	**233**	238	**233**	238	**233**	235
5	**230**	244	**230**	244	**230**	240
5	**230**	240	**230**	240	**230**	233
5	**231**	240	**229**	240	**226**	237
5	**228**	240	**228**	240	**228**	236
5	**232**	239	**228**	239	**228**	236
5	**231**	240	**229**	240	**229**	235
5	**232**	239	**231**	239	**231**	236
$U(\{3,\ldots,7\})$	**226**	231	**226**	231	226	226
$U(\{3,\ldots,7\})$	**228**	231	**226**	231	**226**	229
$U(\{3,\ldots,7\})$	**228**	235	**227**	235	**227**	231
$U(\{3,\ldots,7\})$	**228**	231	**225**	231	**225**	229
$U(\{3,\ldots,7\})$	**231**	235	**229**	235	**227**	230
$U(\{3,\ldots,7\})$	**230**	234	**227**	234	**227**	229
$U(\{3,\ldots,7\})$	**228**	236	**225**	236	**224**	228
$U(\{3,\ldots,7\})$	**226**	234	**225**	234	**225**	230
$U(\{3,\ldots,7\})$	**231**	234	**225**	233	**222**	229
$U(\{3,\ldots,7\})$	**231**	232	**225**	232	**222**	227

Notes. $m = 1{,}000$, $n = 2{,}500$, $\mathbf{c} = \mathbf{e}$. The recorded values denote the objective function, and bold font outlines better outcomes.

Table 2 Results for Weighted Set Covering

	5 hr		10 hr		20 hr	
w	ALG	CPLEX	ALG	CPLEX	ALG	CPLEX
3	**150,453**	151,662	**150,085**	150,691	149,615	**149,390**
3	**150,557**	150,785	**150,110**	150,785	**149,230**	149,833
3	**150,782**	151,130	**150,233**	151,130	149,923	**149,287**
3	151,789	150,917	**150,062**	150,917	**149,486**	150,264
3	**150,449**	151,781	**150,404**	151,709	150,233	**149,756**
3	**149,449**	149,728	**149,449**	149,728	149,449	**148,999**
3	**150,337**	151,202	**150,337**	151,202	150,337	**148,635**
3	**150,088**	151,306	**149,860**	150,740	**149,503**	149,559
3	**149,676**	150,868	**149,609**	150,868	**149,293**	149,403
3	**149,791**	150,524	**149,608**	150,440	**148,703**	149,052
5	**100,264**	107,920	**99,663**	107,920	**99,663**	103,111
5	**102,454**	107,776	**100,943**	107,776	**100,131**	102,393
5	**100,266**	107,190	**99,837**	105,185	**99,837**	100,904
5	**100,393**	106,231	**100,393**	106,231	**100,393**	101,017
5	**100,341**	107,072	**100,180**	107,072	**99,911**	102,651
5	**101,272**	106,268	**100,585**	106,268	**98,989**	101,442
5	**100,718**	107,542	**99,978**	107,542	**99,978**	102,396
5	**101,070**	108,647	**100,530**	108,647	**99,651**	103,266
5	**100,592**	106,986	**100,288**	106,986	**99,970**	103,100
5	**100,084**	108,170	**100,084**	108,170	**100,084**	102,330
$U(\{3,\ldots,7\})$	**99,021**	102,026	**98,803**	102,026	**98,803**	100,951
$U(\{3,\ldots,7\})$	**98,330**	104,147	**98,235**	104,147	**98,235**	100,533
$U(\{3,\ldots,7\})$	**99,630**	101,245	**99,491**	100,789	**98,429**	98,552
$U(\{3,\ldots,7\})$	**99,610**	102,623	**98,765**	102,623	**97,928**	100,997
$U(\{3,\ldots,7\})$	**99,930**	101,656	**99,605**	101,404	**98,801**	99,790
$U(\{3,\ldots,7\})$	**99,485**	102,107	**98,665**	102,104	**98,158**	99,624
$U(\{3,\ldots,7\})$	**99,219**	102,449	**99,056**	100,877	**98,570**	99,128
$U(\{3,\ldots,7\})$	**99,109**	103,281	**98,946**	103,281	**98,905**	100,709
$U(\{3,\ldots,7\})$	**100,868**	102,188	**99,973**	102,188	**99,533**	100,930
$U(\{3,\ldots,7\})$	**100,998**	102,991	**100,285**	102,991	**100,285**	100,837

Notes. $m = 1{,}000$, $n = 2{,}500$, $c_i \in U([400, 500])$. The recorded values denote the objective function, and bold font outlines better outcomes.

ILOG). We note that we have also attempted other settings for *MIP emphasis* in our initial tests (e.g., one, which corresponds to emphasizing feasibility over optimality), but the outcomes were consistently dominated by the "hidden feasibility," which prompted us to remove them from the final results reported here.

Just as with set covering, we find that Algorithm 1 is able to outperform (both runs of) CPLEX in a considerably large number of tests, particularly those corresponding to a fixed w. In all fairness, however, we acknowledge that CPLEX remains a general-purpose solver, searching for provably optimal solutions, no matter which *MIP emphasis* settings are used.

6.3. Mixed Integer Programming Library (MIPLIB) Instances

We have also run our algorithm on three of the instances in the 2003 MIPLIB library (refer to Achterberg et al. 2006 for details). In particular, we focused on the problems *manna81*, *fast0507*, and *seymour*. We note that, whereas the latter two instances directly matched our problem formulation given in Equation (1), *manna81* involved integer knapsack constraints and integer variables, so that we had to adjust our algorithm in order to handle such an instance. Table 6 contains details of the problem structures, as well as a comparison of the optimal value of the

Table 3 Results for Large Instances of Set Covering

	10 hours		20 hours		100 hours		200 hours	
	ALG	CPLEX	ALG	CPLEX	ALG	CPLEX	ALG	CPLEX
Unweighted	969	**904**	**858**	887	**826**	865	**826**	858
Weighted	454,495	**393,540**	432,562	**393,540**	**367,516**	381,087	**366,021**	381,087

Notes. $m = 4{,}000$, $n = 10{,}000$, $\mathbf{c} = \mathbf{e}$, and w generated uniformly at random from $\{3,\ldots,7\}$. Recorded values denote the objective function, and bold outlines better outcomes.

Table 4 Results for Unweighted Set Packing

	5 hr			10 hr			20 hr		
w	ALG	CPX(0)	CPX(4)	ALG	CPX(0)	CPX(4)	ALG	CPX(0)	CPX(4)
3	**322**	319	319	**322**	319	321	**322**	**322**	**322**
3	314	319	**320**	314	319	**322**	321	322	**323**
3	320	321	**322**	320	321	**322**	320	**322**	**322**
3	**320**	318	319	320	320	**321**	320	**321**	**321**
3	**322**	319	320	**323**	320	321	**323**	**323**	322
5	**166**	155	161	**167**	155	162	**167**	155	162
5	**165**	159	159	**168**	159	160	**168**	161	162
5	**165**	160	159	**166**	160	159	**166**	160	161
5	**167**	157	159	**167**	157	160	**167**	157	163
5	**164**	156	158	**166**	156	159	**166**	162	163
7	**104**	95	95	**105**	95	97	**105**	98	100
7	**103**	95	95	**106**	95	95	**106**	95	98
7	**105**	95	95	**105**	95	97	**105**	95	100
7	**105**	97	96	**105**	97	96	**105**	97	99
7	**105**	97	96	**105**	97	96	**105**	97	97
$U(\{3,\ldots,7\})$	255	255	**258**	255	255	**258**	255	258	**259**
$U(\{3,\ldots,7\})$	253	255	**260**	255	255	**261**	255	257	**261**
$U(\{3,\ldots,7\})$	251	**256**	**256**	253	**256**	**256**	253	**258**	256
$U(\{3,\ldots,7\})$	249	252	**255**	249	252	**255**	249	253	**256**
$U(\{3,\ldots,7\})$	256	258	**259**	256	258	**259**	256	**259**	**259**

Notes. $m = 1{,}000$, $n = 2{,}500$, $\mathbf{c} = \mathbf{e}$. Recorded values denote the objective function, and bold outlines better outcomes. CPX(i) denotes a CPLEX run with parameter *MIP emphasis* $= i$.

problem (column OPT) with the results obtained by running Algorithm 1 and CPLEX (columns ALG and CPLEX, respectively). We note that Algorithm 1 was run with an identical parameter setting as described in the debut of §6, and CPLEX was run with *MIP emphasis* set to 4.

As can be noticed from the results in Table 6, our algorithm recovers the optimal value in instance

Table 5 Results for Weighted Set Packing

	5 hr			10 hr			20 hr		
w	ALG	CPX(0)	CPX(4)	ALG	CPX(0)	CPX(4)	ALG	CPX(0)	CPX(4)
3	146,896	147,663	**148,706**	147,800	147,663	**148,894**	148,905	148,617	**149,500**
3	147,796	147,914	**149,511**	147,796	147,914	**149,726**	147,796	149,116	**149,726**
3	147,951	147,479	**148,125**	147,951	147,479	**148,733**	147,951	148,583	**148,788**
3	147,421	147,023	**148,568**	147,421	147,023	**148,889**	147,421	148,399	**148,889**
3	148,545	148,208	**149,104**	148,545	148,869	**149,325**	148,545	149,261	**149,823**
5	**75,937**	71,799	74,865	**76,590**	71,799	75,488	**77,531**	75,594	76,806
5	**76,928**	72,762	73,080	**77,566**	72,762	73,149	**77,566**	74,790	74,752
5	**76,841**	73,447	74,747	**77,681**	73,447	74,747	**77,726**	76,086	75,696
5	**77,475**	72,492	74,392	**77,681**	73,231	74,562	**77,681**	75,810	76,299
5	**76,606**	73,361	73,679	**76,985**	73,361	73,750	**77,674**	75,376	74,995
7	**48,200**	43,708	45,204	**48,200**	43,708	45,577	**48,200**	46,052	46,187
7	**48,559**	43,548	44,001	**48,895**	44,579	45,809	**48,895**	45,486	47,083
7	**48,019**	43,433	44,602	**48,019**	44,658	45,177	**48,019**	46,287	45,606
7	**48,297**	43,667	44,517	**48,297**	43,667	45,272	**48,297**	45,721	47,383
7	**48,721**	44,046	45,280	**48,721**	44,046	46,767	**48,721**	44,641	47,545
$U(\{3,\ldots,7\})$	114,934	113,760	**115,852**	114,934	113,760	**115,852**	114,934	114,948	**115,909**
$U(\{3,\ldots,7\})$	118,242	117,505	**118,261**	**118,587**	117,505	118,344	**118,587**	118,477	118,447
$U(\{3,\ldots,7\})$	116,733	116,896	**118,334**	117,115	116,896	**118,334**	117,115	117,686	**118,334**
$U(\{3,\ldots,7\})$	117,227	116,139	**117,317**	117,227	116,421	**117,317**	117,227	**117,616**	117,317
$U(\{3,\ldots,7\})$	114,864	115,134	**116,356**	114,864	115,134	**116,356**	114,864	115,915	**116,356**

Notes. $m = 1{,}000$, $n = 2{,}500$, $c_i \in U([400, 500])$. Values in bold denote better outcomes. CPX(i) denotes CPLEX run with *MIP emphasis* $= i$.

Table 6 Instances in MIPLIB Library and Comparison of the Results Obtained with ALG and CPLEX

			Constraints					
Problem name	Rows	Columns	Pack	Cover	Knapsack	OPT	ALG	CPLEX
manna81	6,480	3,321	0	0	6,480	−13,164	−13,164	−13,164
fast0507	507	63,009	3	504	0	174	188	174
seymour	4,944	1,372	285	4,659	0	423	444	423

Note. OPT denotes the optimal value of the problem.

manna81, but stops short of doing the same for *fast0507* and *seymour*, with a relative optimality gap of 8.04% and 4.96%, respectively. We suspect that for these instances (particularly *seymour*, which also causes problems for CPLEX) one would need to allow a considerably larger neighborhood for the local search procedure (e.g., by a larger value of the parameter Q), in order to recover the true optimum.

7. Conclusions

In this paper, we have presented a new class of general-purpose heuristic methods for solving large, sparse binary optimization problems. The formulation of the central algorithm, based on the notion of *interesting* solutions and their *traces*, provides flexibility in terms of the exact implementation, and allows the user to directly influence the complexity-performance trade-off through the adjustable parameter Q.

In addition to interesting theoretical properties (pseudo-polynomial running times and performance guarantees), we feel that the proposed method has practical promise, as it is applicable in fairly general settings, and it is competitive with leading optimization packages in computational tests on fairly large instances of randomly generated set packing and set covering problems.

Electronic Companion

An electronic companion to this paper is available as part of the online version at http://dx.doi.org/10.1287/ijoc.1110.0496.

Acknowledgments

The authors thank Jingting Zhou for research assistance with the computational study, and the editor-in-chief, the area editor, the associate editor and two anonymous referees, whose comments and suggestions have improved the content and exposition of the present paper.

References

Aarts, E., J. K. Lenstra, eds. 1997. *Local Search in Combinatorial Optimization*. John Wiley & Sons, New York.

Achterberg, T., T. Koch, A. Martin. 2006. MIPLIB 2003. *Oper. Res. Lett.* **34**(4) 361–372.

Aickelin, U. 2002. An indirect genetic algorithm for set covering problems. *J. Oper. Res. Society* **53**(10) 1118–1126.

Al-Sultan, K. S., M. F. Hussain, J. S. Nizami. 1996. A genetic algorithm for the set covering problem. *J. Oper. Res. Society* **47**(5) 702–709.

Arkin, E. M., R. Hassin. 1998. On local search for weighted k-set packing. *Math. Oper. Res.* **23**(1) 640–648.

Bakhtiari, S., R. Safavi-Naini, J. Pieprzyk. 1995. Cryptographic hash functions: A survey. Technical Report 95-09, Department of Computer Science, University of Wollongong.

Balas, E., A. Ho. 1980. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Combinatorial Optimization*. Mathematical Programming Studies, Vol. 12. Springer, Berlin, Heidelberg, 37–60.

Balas, E., C.. H. Martin. 1980. Pivot and complement–a heuristic for 0–1 programming. *Management Sci.* **26**(1) 86–96.

Balas, E., S. Ceria, G. Cornuéjols. 1993. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Math. Programming* **58**(1-3) 295–324.

Beasley, J. E. 1990. A Lagrangian heuristic for set-covering problems. *Naval Res. Logist.* **37**(1) 151–164.

Caprara, A., M. Fischetti, P. Toth. 1999. A heuristic method for the set covering problem. *Oper. Res.* **47**(5) 730–743.

Caprara, A., P. Toth, M. Fischetti. 2000. Algorithms for the set covering problem. *Annals Oper. Res.* **98**(1) 353–371.

Chu, P. C., J. E. Beasley. 1996. A genetic algorithm for the set covering problem. *Eur. J. Oper. Res.* **94**(2) 392–404.

Cormen, T. H., C. Stein, R. L. Rivest, C. E. Leiserson. 2001. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Eckstein, J., M. Nediak. 2007. Pivot, cut, and dive: A heuristic for 0–1 mixed integer programming. *J. Heuristics* **13**(5) 471–503.

Fischetti, M., F. Glover, A. Lodi. 2005. The feasibility pump. *Math. Programming* **104**(1) 91–104.

Hurkens, C. A. J., A. Schrijver. 1989. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discret. Math.* **2**(1) 68–72.

Jia, W., C. Zhang, J. Chen. 2004. An efficient parameterized algorithm for m-set packing. *J. Algorithms* **50**(1) 106–117.

Koutis, I. 2005. A faster parameterized algorithm for set packing. *Inform. Processing Lett.* **94**(1) 7–9.