# Convex optimization under combinatorial sparsity constraints*

Christoph Buchheim[1] and Emiliano Traversi[2]

[1] Fakultät für Mathematik, TU Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany, christoph.buchheim@tu-dortmund.de
[2] LIPN, Université Paris 13, 99 Avenue Jean-Baptiste Clément, 93430 Villetaneuse, France, emiliano.traversi@lipn.univ-paris13.fr

**Abstract.** We present a heuristic approach for convex optimization problems containing sparsity constraints. The latter can be cardinality constraints, but our approach also covers more complex constraints on the support of the solution. For the special case that the support is required to belong to a matroid, we propose an exchange heuristic adapting the support in every iteration. The entering non-zero is determined by considering the dual of the given convex problem where the variables not belonging to the current support are fixed to zero. While this algorithm is purely heuristic, we show experimentally that it often finds solutions very close to the optimal ones in the case of the cardinality-constrained knapsack problem and for the sparse regression problem.

**Keywords:** Sparse Optimization · Cardinality Constrained Knapsack · Sparse Regression · Matroids

## 1 Introduction

We consider the problem of finding sparse solutions to a convex optimization problem of the general form

$$\begin{array}{rl} \min & f(x) \\ \text{s.t.} & x \in C \ , \end{array} \tag{1}$$

where $C \subseteq \mathbb{R}^n$ is a convex set and $f\colon C \to \mathbb{R}$ is a convex function. Our notion of sparsity is given by a combinatorial set $T \subseteq \{0,1\}^n$ containing the feasible supports. In the following, we assume w.l.o.g. that $T$ is hereditary, i.e., if $t \in T$ and $t' \leq t$, we also have $t' \in T$. Throughout this paper, we only assume that $T$ is accessible via a membership and a linear optimization oracle. We are particularly interested in the case where $T$ is a matroid, which directly generalizes the case of an ordinary cardinality contraint of the type $||x||_0 \leq k$.

* This work was partially supported by the German Research Foundation (DFG) under grant no. BU 2313/4.

The problem addressed can thus be written as

$$\begin{array}{rll} \min & f(x) & \\ \text{s.t.} & x \in C & \\ & x_i = 0 & \text{if } t_i = 0 \\ & t \in T\ . & \end{array} \tag{2}$$

Note that we optimize over both $x$ and $t$ here. Not surprisingly, Problem (2) is NP-hard. This has been shown by de Farias and Nemhauser [6] in the case of the continuous knapsack problem subject to cardinality constraints. The second constraint in (2) can be replaced by complementarity constraints $x_i(1 - t_i) = 0$ for $i = 1, \ldots, n$. Problem (2) is equivalent to finding a $t \in T$ that minimizes

$$\begin{array}{rrll} f_t^* & := & \min & f(x) \\ & & \text{s.t.} & x \in C \\ & & & x_i = 0 \ \text{ if } t_i = 0\ . \end{array} \tag{3}$$

For sake of simplicity, we restrict ourselves to cases where Problem (3) is feasible for any given $t \in T$. Our proposed heuristic for solving Problem (2) iteratively adapts the support $t$. In the general case, we propose to compute a new support $t' \in T$ by optimizing a certain function over $T$, where the coefficient for each zero entry $t_i$ in the last solution is calculated based on the dual solution of (3). In the case that $T$ is a matroid, we can also consider iterations where only one pair of entries in $t$ is swapped. Our experimental results presented in this paper show that the latter heuristic is capable of finding near-optimal solutions in short running time for a variety of sparse optimization problems.

When $T$ is a uniform matroid, we thus search for cardinality constrained solutions of Problem (1). This has many practical applications, e.g., in Sparse Regression, where we search for a sparse near-solution of a linear system of equations $Ax = b$ [11, 13]. Recently, the first approaches based on Mixed Integer Non-Linear Programming have been proposed to solve this problem exactly [1, 8]. Another important class of applications arises in Portfolio Optimization, where sparse investments are desired [3, 7, 9].

Apart from the case where $T$ is a uniform matroid (or a partition matroid) modeling (partial) cardinality constraints, our approach is motivated by another application: we are interested in finding a symmetric matrix $X$ such that $Q \succcurlyeq X$ for a given symmetric matrix $Q$ and such that the non-zero entries in $X$ form a forest in the complete graph indexed by the rows (or columns) of $X$. Given such $X$, an underestimator of the quadratic binary optimization problem

$$\begin{array}{rl} \min & x^\top Q x \\ \text{s.t.} & x \in \{0,1\}^n \end{array}$$

is obtained by replacing $Q$ by $X$ and then solving the resulting sparse problem, which can be done in polynomial time. The optimal value of the latter problem is then a lower bound for the original problem, which can be used to strengthen the bounds obtained from separable underestimators within the branch-and-bound algorithm presented in [4].

This paper is organized as follows. In Section 2, we give some theoretical background motivating our approach. The primal heuristic is devised in Section 3, both for the matroid case and for the general case. Finally, the results of an experimental evaluation are presented in Section 4. They show that the heuristic is capable of finding near-optimal solutions in short running time for various types of conic optimization problems, where we consider three different underlying cones: the non-negative orthant, the cone of semidefinite matrices, and the second-order cone. Section 5 concludes.

## 2 Motivation and preliminaries

The crucial step in our algorithm is to decide which fixings $x_i = 0$ should be released in the next iteration, and which variables should be fixed instead. For the first task, we use dual information. The idea is that the dual variable for the constraint $x_i = 0$ tells us how promising it is to relax the constraint.

### 2.1 General convex case

More precisely, in the general case, we consider the partial dual of Problem (3) with respect to the current fixings,

$$\max_{d \in \mathbb{R}^{\bar{t}}} \min_{x \in C} f(x) + d^\top x_{\bar{t}} \, , \tag{4}$$

where $\mathbb{R}^{\bar{t}}$ denotes the subspace of $\mathbb{R}^n$ given by the dimensions $i$ with $t_i = 0$ and $x_{\bar{t}}$ is the vector $x$ restricted to these dimensions. Now taking the optimal solution $d^*$ of Problem (4), the entry $d_i^*$ should give a hint about how much improvement can be expected from releasing variable $x_i$.

However, there is a problem with this approach, as the dual solution in general is not unique. To illustrate this, consider the case of a linear problem

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \\ & x_i = 0 \ \text{ if } t_i = 0 \, . \end{aligned} \tag{5}$$

Then Problem (4) reads

$$\max_{d \in \mathbb{R}^{\bar{t}}} \min_{\substack{Ax=b \\ x \geq 0}} c^\top x + d^\top x_{\bar{t}} \;=\; \max_{d \in \mathbb{R}^{\bar{t}}} \max_{\substack{d+(c-A^\top y)_{\bar{t}} \geq 0 \\ (c-A^\top y)_t \geq 0}} b^\top y \, ,$$

so that every $d \geq (A^\top y^* - c)_{\bar{t}}$ is an optimal solution of (4) if $y^*$ is an optimal dual multiplier for $Ax = b$. Consequently, using an arbitrary dual solution does not give any useful information.

Instead, we propose to start from a slightly different model. We consider the problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in C \\ & ||x_{\bar{t}}|| \leq \varepsilon \end{aligned} \tag{6}$$

for $\varepsilon > 0$ and an arbitrary norm $||\cdot||$. Then the partial dual with respect to the fixings reads

$$\max_{d\in\mathbb{R}^{\bar{t}}} \left( -\varepsilon||d||^* + \min_{x\in C} \; f(x) + d^\top x_{\bar{t}} \right) , \tag{7}$$

where $||\cdot||^*$ denotes the dual norm to $||\cdot||$. For $\varepsilon \to 0$, this becomes a lexicographic optimization problem, and its optimal solution $d^*$ is obtained by minimizing $||d||^*$ over the optimal set of (4),

$$\arg\max_{d\in\mathbb{R}^{\bar{t}}} \left( \min_{x\in C} f(x) + d^\top x_{\bar{t}} \right) .$$

Choosing the two-norm $||\cdot||_2$, the solution $d^*$ is thus unique. The resulting $d^*$ is used in the heuristic we present in Section 3.

### 2.2 Conic case

In the case of a conic optimization problem, the above calculations can be made more explicit. Consider the problem

$$\begin{array}{rl} \min & c^\top x \\ \text{s.t.} & Ax = b \\ & x \in K \\ & ||x_{\bar{t}}|| \leq \varepsilon \end{array} \tag{8}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $K \subseteq \mathbb{R}^n$ is a closed convex cone. We can then fully dualize in (7) and obtain

$$\begin{array}{rl} \max & b^\top y - \varepsilon||d||^* \\ \text{s.t.} & \binom{c_t - A_t^\top y}{c_{\bar{t}} - A_{\bar{t}}^\top y + d} \in K^* \\ & y \in \mathbb{R}^m \\ & d \in \mathbb{R}^{\bar{t}} \, . \end{array} \tag{9}$$

We then first maximize $b^\top y$ and subsequently calculate the best feasible corresponding $d^*$, i.e., one that minimizes $||d||^*$. When optimizing only the first term of the objective function of (9), we can project out the $d$ variables and solve the following problem

$$\begin{array}{rl} \max & b^\top y \\ \text{s.t.} & c_t - A_t^\top y \in \text{proj}_{\,t} K^* \\ & y \in \mathbb{R}^m \, . \end{array} \tag{10}$$

Here $A_t$ denotes the submatrix of $A$ consisting of columns $i$ with $t_i = 1$. The dual of (10) is

$$\begin{array}{rl} \min & c_t^\top x_t \\ \text{s.t.} & A_t x_t = b \\ & x_t \in (\text{proj}_{\,t} K^*)^* \, , \end{array}$$

which is essentially just Problem (8) where the fixed variables $x_{\bar{t}}$ are eliminated; note however that $(\text{proj}_{\,t} K^*)^* \neq \text{proj}_{\,t} K$ in general. This implies that a set of optimal $y^*$ variables can be obtained by solving the smaller Problem (10) directly. Given $y^*$, the problem of finding the corresponding $d^*$ reduces to

$$\begin{aligned} \min \quad & ||d||^* \\ \text{s.t.} \quad & \binom{c_t - A_t^\top y^*}{c_{\bar t} - A_{\bar t}^\top y^* + d} \in K^* \\ & d \in \mathbb{R}^{\bar t} \,. \end{aligned} \tag{11}$$

Setting $r := c - A^\top y^*$, we thus need to project the point $r_{\bar t}$ to the convex set $\{z \in \mathbb{R}^{\bar t} \mid \binom{r_t}{z} \in K^*\}$, in case $r \notin K^*$. The projection depends on the cone $K$ and possibly on the norm $||\cdot||$. If $K = \mathbb{R}^n_+$ is the non-negative cone, i.e., we deal with linear programs, we have $\mathrm{proj}_{\,t} K^* = \mathbb{R}^t_+$, so that $y^*$ can be computed by solving

$$\begin{aligned} \max \quad & b^\top y \\ \text{s.t.} \quad & c_t - A_t^\top y \geq 0 \end{aligned}$$

and $d^*$ by solving

$$\begin{aligned} \min \quad & ||d||^* \\ \text{s.t.} \quad & d \geq A_{\bar t}^\top y^* - c_{\bar t} \,, \end{aligned}$$

which just means setting $d^* := \max\{0, A_{\bar t}^\top y^* - c_{\bar t}\}$ componentwise.

*Example 1.* Consider

$$\begin{aligned} \min \quad & x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + x_2 = 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

where first $x_1$ is fixed to zero, i.e., $t = (0,1)^\top$. Then (10) reads

$$\begin{aligned} \max \quad & y \\ \text{s.t.} \quad & 2 - y \geq 0 \end{aligned}$$

so that $y^* = 2$ and $A_{\bar t}^\top y^* - c_{\bar t} = 1$. In (11) we thus obtain $d^* = 1$. If instead we fix $x_2$, we obtain $y^* = 1$ and $d^* = 0$. This reflects the fact that releasing $x_2$ does not imply any improvement in the optimal solution, while releasing $x_1$ improves the optimal value even if $x_2$ is fixed instead.

## 3 Exchange heuristic

Following the ideas developed in the last section, the outline of the proposed heuristic for Problem (2) is given below.

1. Set $i := 0$ and choose $t^{(0)} \in T$.
2. Solve (6) (for small $\varepsilon > 0$ or $\varepsilon \to 0$) to obtain $x^*$ and (7) to obtain $d^*$.
3. Calculate $t^{(i+1)}$, based on $x^*$ and $d^*$.
4. If $t^{(i+1)} \neq t^{(k)}$ for all $k \leq i$, set $i := i + 1$ and go to 2.

In order to describe this heuristic algorithm in more detail, it remains to specify the choice of an initial solution (Step 1) and, most importantly, the update rule for the support (Step 3). For the update of $t^{(i)}$ to $t^{(i+1)}$, we distinguish between the matroid and the general case.

*General case update (GU).* For general $T$, we obtain $t^{(i+1)}$ as optimal solution to the following combinatorial optimization problem

$$\max_{t\in T}\ \left(\sum_{t_j^{(i)}=0}|d_j^*|t_j+\sum_{t_j^{(i)}=1}|x_j^*|t_j\right)$$

using the linear optimization oracle. The intuition behind the proposed rule is to use the dual value $d_j^*$ of the fixing of variable $x_j$ as an estimate of the potential contribution to the objective function, if the corresponding fixing is removed. Such potential contribution must be compared with the contribution of each free variable $x_j$, which we simply estimate as $|x_j^*|$. For ensuring that the ranges are comparable, we first scale the vectors $d^*$ and $x^*$ to obtain the same norm.

*Matroid case update (MU).* In the case that $T$ is a matroid, we propose an alternative method, which turns out to outperform the previous approach significantly both in terms of quality and running time; see Section 4 for experimental results. The motivation of this method is the same as above, but it exploits the matroid property, namely, that a solution $t$ can be updated by an element-wise exchange. We first find the index to free as

$$\bar{k} := \operatorname*{arg\,max}_{k:\ t_k^{(i)}=0} |d_k^*|$$

and then we complete the support by computing the variable to fix as

$$\bar{l} := \operatorname*{arg\,min}_{\substack{l:\ t_l^{(i)}=1\\ t^{(i)}+\bar{k}-l\in T}} |x_l^*|\ .$$

The final support is $t^{(i+1)} := t^{(i)} + \bar{k} - \bar{l}$. The assumption that $T$ is a matroid guarantees that there exists at least one candidate for $\bar{l}$ different from $\bar{k}$ in the latter minimization. One advantage of MU in comparison with GU is that we do not need an optimization oracle for $T$ but only a membership oracle.

*Optimal 2-opt update (2OU).* For comparison, still in the matroid case, we also consider the straightforward 2-opt update rule in our experiments. Here, we enumerate all pairs $\bar{k}, \bar{l}$ such that $t^{(i)} + \bar{k} - \bar{l} \in T$ and choose the one leading to the smallest value $f^*_{t^{(i+1)}}$ in the next iteration.

*Initial solution.* For the choice of $t^{(0)}$, different strategies can be considered. A natural choice is to maximize $c^\top t$ over $T$, which can be done by calling the linear optimization oracle for $T$ that we assume to have at hand. Another strategy would be to fix all variables in the problem to zero, to consider dual variables $d^*$ as above, and then maximize $|d^*|^\top t$ over $T$. Both methods lead to a small number of iterations in general. However, in our experiments, it turned out that the quality of the final solution can be improved by starting with a support $t^{(0)}$ that is far from being the optimal one. This can probably be explained by the fact that this avoids to send the algorithm to a local optimum immediately.

## 4 Experimental results

In this section, we provide an experimental evaluation of our approach based on test instances from different sparse optimization problems: the *Cardinality Constrained Continuous Knapsack Problem (CCKP)*, *Sparse Regression (SR)*, and the *Tree Underestimator Problem for Binary Quadratic Problems (TUP)*. The aim of this section is three-fold:

(a) In the tests concerning the CCKP, we show that our heuristic procedure is able to provide almost-optimal solutions within a very short amount of time.
(b) For SR instances, we show that the approach yields a significantly better approximation than the standard Lasso approach, which replaces the cardinality by the one-norm. Even compared to the more sophisticated SparseNet algorithm, we often find better results.
(c) Finally, we further motivate our proposed concept of sparsity. Using spanning trees as feasible supports, the TUP generalizes the method proposed in [4] to efficiently solve combinatorial problems with a quadratic objective function.

For assessing the performance of our algorithm, we use CPLEX 12.6 [5] with an optimality tolerance of $10^{-6}$. The second constraint in Problem (2) is then modeled as a special ordered set (SOS) constraint. For CCKP and SR, we also use CPLEX for solving the convex problems (6) as needed in Step 2 of the algorithm. Dual solutions are also provided by CPLEX. For TUP, we use the SDP solver CSDP [2]. All experiments were carried out on Intel Xeon processors running at 2.50 GHz.

### 4.1 Cardinality constrained continuous knapsack problem

The CCKP is a continuous ($m$-dimensional) knapsack problem where at most $k$ variables are allowed to be strictly greater than zero, for some given integer $k$. It has been introduced and investigated in [6]. As $T$ is the uniform matroid here, we can use all update rules presented in Section 4. We use two sets of instances: the *De Farias et al.* instances are generated as described in [6]. They contain up to 8000 variables ($n$) and 70 knapsack constraints ($m$), the values of $k$ and the densities of the non-zero entries of the knapsack constraints ($d$) were chosen with the objective to make the instances computationally difficult. The *Random* instances are produced in the same way, using knapsack constraints having a density $d$ of 100%, while various cardinalities $k$ are considered, all in the (small) range where the resulting problems turn out to be non-trivial.

We first compare the performances of our different update rules, namely, the matroid exchange `MU`, the general exchange `GU`, and the optimal pairwise exchange `2OU`. Table 1 shows results for random instances with $n = 300$, stating averages over 10 instances in each line. For each heuristic approach, we show the best primal bound obtained (`best`), the computation time in seconds (`time`), and the number of iterations (`iter`). The values of the primal bounds are normalized by dividing them by the best of the three bounds.

As expected, the optimal pairwise exchange provides the best primal solution for each instance. However, it is several orders of magnitude slower in terms of computing time. For this reason, the optimal pairwise exchange is not suitable if we want to tackle bigger instances. Moreover, the difference in the solution quality is very small: it is below 1 % for the matroid exchange and below 6 % for the general exchange.

| | | | MU | | | GU | | | 2OU | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | best | time | iter | best | time | iter | best | time | iter |
| 300 | 1 | 90 | 1.000 | 0.00 | 6.8 | 0.991 | 0.01 | 8.1 | 1.000 | 32.00 | 10.2 |
| 300 | 1 | 120 | 0.999 | 0.01 | 36.8 | 0.945 | 0.03 | 22.3 | 1.000 | 90.22 | 38.4 |
| 300 | 1 | 150 | 1.000 | 0.01 | 45.1 | 1.000 | 0.01 | 7.7 | 1.000 | 75.06 | 46.2 |
| 300 | 10 | 90 | 0.994 | 0.02 | 13.2 | 0.966 | 0.03 | 11.6 | 1.000 | 304.30 | 19.1 |
| 300 | 10 | 120 | 1.000 | 0.01 | 17.2 | 1.000 | 0.23 | 71.0 | 1.000 | 116.13 | 19.1 |
| 300 | 10 | 150 | 1.000 | 0.01 | 9.9 | 1.000 | 0.01 | 9.5 | 1.000 | 257.69 | 52.3 |
| 300 | 100 | 90 | 0.998 | 0.09 | 17.3 | 0.978 | 0.21 | 15.8 | 1.000 | 1313.59 | 17.5 |
| 300 | 100 | 120 | 1.000 | 0.04 | 13.6 | 1.000 | 0.09 | 13.4 | 1.000 | 643.06 | 17.8 |
| 300 | 100 | 150 | 1.000 | 0.04 | 6.5 | 1.000 | 0.08 | 10.3 | 1.000 | 2621.16 | 73.4 |

**Table 1.** Results for random instances of CCKP, comparing matroid exchange (`MU`), general exchange (`GU`), and 2-opt heuristics (`2OU`)

When comparing the general exchange rule with the matroid exchange, it turns out that the matroid rule yields better (or equally good) results on all instances, while often using more iterations and the same running time on average. This trend is confirmed by the results for large random instances presented in Table 2, but here the matroid exchange rule even outperforms the general exchange rule in terms of running time. For this reason, we use the matroid exchange rule in the following experiments. Note that the running time of all three heuristics is dominated by the time needed to solve (13) for fixed $t$ using CPLEX.

In Table 3 we present the results concerning the *de Farias et al.* instances. We compare the performance of our heuristic with the performance of the commercial solver CPLEX. The table first presents the time, the primal bound, and the number of iteration of the heuristic (`best`, `time` and `iter`), and then the best primal and dual bounds and the computing time needed by CPLEX (`primal`, `dual`, and `time`). We set a time limit of one cpu hour. The results show that the heuristic – in few iterations and seconds – provides an optimal solution 5 times out of 16. In the remaining cases, the heuristic solution is at most 0.53% away from the dual bound of CPLEX and hence from optimality.

To assess the solution quality of our heuristic against the optimal solution on a larger set of instances, we again consider random instances and solve them to optimality by CPLEX, now without a time limit; see Table 4. Here, the value `%gap` states how far our heuristic solution is from optimality. While some of the instances are very hard to solve by CPLEX, our heuristic can always find a solution within 0.91 % from optimality in a running time that on average never exceeds 0.03 seconds for $n = 200$ and 0.1 seconds for $n = 400$. The table shows

| | | | MU | | | GU | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | best | time | iter | best | time | iter |
| 6000 | 10 | 1800 | 1.000 | 15.3 | 37.3 | 0.995 | 32.6 | 16.2 |
| 6000 | 10 | 2400 | 1.000 | 3.4 | 172.0 | 1.000 | 6.3 | 11.0 |
| 6000 | 10 | 3000 | 1.000 | 2.5 | 53.0 | 1.000 | 5.2 | 9.8 |
| 6000 | 100 | 1800 | 1.000 | 56.7 | 76.0 | 0.992 | 103.9 | 18.0 |
| 6000 | 100 | 2400 | 1.000 | 14.9 | 21.9 | 1.000 | 58.6 | 16.5 |
| 6000 | 100 | 3000 | 1.000 | 13.5 | 4.7 | 1.000 | 48.1 | 10.1 |
| 6000 | 1000 | 1800 | 1.000 | 293.1 | 103.0 | 0.995 | 591.4 | 17.7 |
| 6000 | 1000 | 2400 | 1.000 | 100.4 | 10.9 | 1.000 | 290.4 | 17.9 |
| 6000 | 1000 | 3000 | 1.000 | 96.9 | 4.7 | 1.000 | 243.6 | 10.4 |
| 8000 | 10 | 2400 | 1.000 | 36.0 | 50.6 | 0.994 | 59.4 | 16.5 |
| 8000 | 10 | 3200 | 1.000 | 6.3 | 232.2 | 1.000 | 11.0 | 9.5 |
| 8000 | 10 | 4000 | 1.000 | 4.5 | 70.4 | 1.000 | 9.4 | 9.5 |
| 8000 | 100 | 2400 | 1.000 | 120.3 | 93.6 | 0.993 | 191.7 | 19.8 |
| 8000 | 100 | 3200 | 1.000 | 28.2 | 23.7 | 1.000 | 117.1 | 16.3 |
| 8000 | 100 | 4000 | 1.000 | 30.6 | 4.9 | 1.000 | 106.6 | 10.9 |
| 8000 | 1000 | 2400 | 1.000 | 607.8 | 141.1 | 0.996 | 869.8 | 15.6 |
| 8000 | 1000 | 3200 | 1.000 | 157.2 | 9.6 | 1.000 | 466.2 | 17.7 |
| 8000 | 1000 | 4000 | 1.000 | 159.1 | 4.6 | 1.000 | 398.8 | 11.2 |

**Table 2.** Results for large random instances of CCKP, comparing matroid exchange (MU) and general exchange (GU)

| | | | | MU | | | CPLEX | | |
|---|---|---|---|---|---|---|---|---|---|
| $m$ | $n$ | $k$ | $d$ | best | time | iter | primal | dual | time |
| 20 | 500 | 150 | 50 | 3384.2 | 0.130 | 25 | 3401.0 | 3401.0 | 214.3 |
| 20 | 1000 | 300 | 50 | 6830.9 | 0.630 | 41 | 6861.0 | 6861.0 | 264.2 |
| 20 | 1500 | 450 | 50 | 10295.7 | 0.740 | 28 | 10331.0 | 10331.7 | TL |
| 20 | 2000 | 600 | 30 | 13791.4 | 0.680 | 25 | 13833.0 | 13833.4 | TL |
| 20 | 2500 | 750 | 42 | 17296.3 | 0.650 | 13 | 17326.0 | 17326.0 | 0.9 |
| 30 | 3000 | 1000 | 33 | 22352.9 | 6.500 | 136 | 22425.6 | 22438.0 | TL |
| 30 | 3500 | 1000 | 28 | 23211.8 | 0.680 | 11 | 23220.0 | 23220.0 | 1.6 |
| 50 | 4000 | 1000 | 25 | 23491.0 | 0.110 | 5 | 23491.0 | 23491.0 | 3.0 |
| 50 | 4500 | 2000 | 30 | 34021.5 | 8.160 | 357 | 34021.5 | 34021.5 | 0.7 |
| 50 | 5000 | 2000 | 20 | 39577.3 | 25.910 | 661 | 39577.3 | 39577.3 | 0.8 |
| 50 | 5500 | 2000 | 18 | 43377.1 | 44.880 | 389 | 43513.4 | 43547.1 | TL |
| 50 | 6000 | 2000 | 16 | 44999.3 | 40.380 | 192 | 45214.6 | 45238.1 | TL |
| 50 | 6500 | 1000 | 15 | 24221.0 | 0.220 | 5 | 24221.0 | 24221.0 | 5.9 |
| 50 | 7000 | 2000 | 14 | 46353.8 | 2.830 | 31 | 46388.0 | 46388.0 | 7.4 |
| 70 | 7500 | 2000 | 13 | 46663.0 | 0.520 | 8 | 46669.0 | 46669.0 | 11.8 |
| 70 | 8000 | 3000 | 25 | 60202.5 | 91.550 | 808 | 60202.5 | 60202.5 | 1.8 |

**Table 3.** *De Farias et al.* instances, matroid exchange vs. CPLEX

that instances with a very small or large $k$ tend to be easier to solve for both CPLEX and MU. This is supposedly due to the fact that for large values of $k$, the cardinality constraint tends to be redundant, while for very small values of $k$, the number of feasible solutions is limited.

| | | | MU | | | CPLEX | |
|---|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | %gap | time | iter | time | nodes |
| 200 | 1 | 60 | 0.09 | 0.000 | 7.1 | 0.0 | 91.0 |
| 200 | 2 | 60 | 0.14 | 0.000 | 8.5 | 0.0 | 41.7 |
| 200 | 5 | 60 | 0.51 | 0.006 | 10.2 | 0.3 | 1182.2 |
| 200 | 10 | 60 | 0.53 | 0.018 | 17.8 | 11.4 | 43317.5 |
| 200 | 20 | 60 | 0.91 | 0.018 | 15.0 | 9.3 | 23404.3 |
| 200 | 50 | 60 | 0.64 | 0.030 | 15.5 | 139.4 | 96191.6 |
| 200 | 1 | 70 | 0.44 | 0.005 | 18.4 | 0.1 | 216.8 |
| 200 | 2 | 70 | 0.44 | 0.005 | 21.8 | 0.2 | 673.7 |
| 200 | 5 | 70 | 0.00 | 0.009 | 26.4 | 0.0 | 0.0 |
| 200 | 10 | 70 | 0.00 | 0.010 | 24.3 | 0.0 | 0.0 |
| 200 | 20 | 70 | 0.00 | 0.011 | 22.4 | 0.0 | 3.0 |
| 200 | 50 | 70 | 0.00 | 0.018 | 20.2 | 0.1 | 58.1 |
| 400 | 1 | 120 | 0.07 | 0.005 | 8.1 | 0.0 | 68.0 |
| 400 | 2 | 120 | 0.12 | 0.008 | 9.8 | 0.1 | 64.8 |
| 400 | 5 | 120 | 0.22 | 0.027 | 14.1 | 0.9 | 2120.2 |
| 400 | 10 | 120 | 0.45 | 0.055 | 18.9 | 425.5 | 26607.7 |
| 400 | 20 | 120 | 0.53 | 0.102 | 23.6 | 1482.7 | 64157.4 |
| 400 | 50 | 120 | 0.46 | 0.099 | 19.5 | 3640.5 | 2.7 |
| 400 | 1 | 140 | 0.43 | 0.023 | 26.1 | 0.1 | 258.6 |
| 400 | 2 | 140 | 0.34 | 0.025 | 35.0 | 3.2 | 8681.8 |
| 400 | 5 | 140 | 0.00 | 0.027 | 45.2 | 0.0 | 0.0 |
| 400 | 10 | 140 | 0.00 | 0.030 | 38.5 | 0.0 | 0.0 |
| 400 | 20 | 140 | 0.00 | 0.038 | 32.8 | 0.0 | 0.0 |
| 400 | 50 | 140 | 0.00 | 0.058 | 28.6 | 0.0 | 0.0 |

**Table 4.** Random instances of CCKP, comparison with CPLEX

### 4.2 Sparse Regression

In sparse (linear) regression, one searches for a sparse (approximate) solution of a system of linear equations $Ax = b$. This has numerous applications, e.g., in signal processing. We model the problem as follows: we minimize $||Ax - b||_2$ subject to the cardinality constraint $||x||_0 \leq k$, where the matrix $A \in \mathbb{R}^{m \times n}$ and the vector $b \in \mathbb{R}^m$ are generated as proposed in [13]: for a given number $n$ of variables, we first produce a matrix $A \in \mathbb{R}^{(0.5n) \times n}$, with entries randomly generated using a standard normal distribution. Then we generate a random vector $x_0 \in \mathbb{R}^n$ with 90% random entries fixed to zero and the remaining 10% entries again chosen according to a standard normal distribution. Finally, we set $b := Ax_0$, we thus know that the optimal solution of the above problem is zero when setting $k = \frac{n}{10}$ and can investigate the quality of the solutions obtained by our heuristic method. In Table 5, we state the results, where every line represents 10 different instances with the stated parameters $n$, $m = \frac{n}{2}$, and $k = \frac{n}{10}$. We list the average value $||Ax^* - b||_2$ of the best found solution $x^*$ (`best`) and the number of instances where the heuristic found an optimal solution (`#opt`). Moreover, we report the average running time (`time`) and the average number of iterations (`iter`) used by our exchange heuristic.

| | | | MU | | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $k$ | best | #opt | time | iter |
| 200 | 100 | 20 | 0.025 | 2/10 | 0.5 | 22.8 |
| 400 | 200 | 40 | 0.019 | 1/10 | 3.8 | 41.7 |
| 600 | 300 | 60 | 0.001 | 6/10 | 17.4 | 61.3 |
| 800 | 400 | 80 | 0.021 | 0/10 | 52.9 | 81.3 |
| 1000 | 500 | 100 | 0.073 | 2/10 | 108.2 | 100.9 |
| 1200 | 600 | 120 | 0.161 | 2/10 | 204.7 | 120.5 |
| 1400 | 700 | 140 | 0.082 | 0/10 | 372.2 | 142.7 |
| 1600 | 800 | 160 | 0.408 | 3/10 | 631.4 | 162.1 |
| 1800 | 900 | 180 | 0.052 | 3/10 | 887.5 | 161.6 |
| 2000 | 1000 | 200 | 0.158 | 0/10 | 1802.5 | 206.5 |

**Table 5.** Results for random instances of SR, matroid exchange heuristics

It turns out that the average value of the best solution is always very close to zero, i.e., to the global optimum. Even for larger instances, the heuristic finds an optimal solution in some cases. The average number of iterations is slightly above $k$ for all instance sizes, while running times increase significantly for larger $n$. However, even for $n = 2000$, the running time is below 25 cpu minutes, while solutions are very close to being optimal.

We next performed a second set of experiments. Here we used the same instances as above except that, as in [13], we perturb all entries of $b$ by a tenth of a value that is produced using a standard normal distribution. We compare our approach to the well-known Lasso approach to sparse optimization, consisting in replacing the constraint $||x||_0 \leq k$ by the convex constraint $||x||_1 \leq \tau$ for an appropriate $\tau$. For the comparison, we first minimize $||Ax - b||_2$ subject to $||x||_1 \leq \frac{1}{20}n$. This is a convex problem which we can solve using CPLEX. The choice of $\tau = \frac{1}{20}n$ is motivated by the fact that the resulting cardinality is roughly $\frac{1}{10}n$ again. Then we fix the cardinality $k$ of the resulting vector and apply our exchange heuristic to the same instance subject to $||x||_0 \leq k$. The main objective is to show that, even if Lasso can choose the cardinality, our heuristic finds significantly better results with the same sparsity.

Results are presented in Table 6. Again, for each size, we report averages over 10 instances. For both approaches, we report the average objective value of the best solution found (`best`). Moreover, we show the cardinality $k$ of the solution obtained by the Lasso approach, which is used for the cardinality constraint in the matroid exchange heuristic. The latter is used as described above (`MU`). We also report the time needed by `MU` and the one-norm of the best solution it computes (`norm`).

| | | Lasso | | MU | | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | best | $k$ | best | time | norm |
| 200 | 100 | 13.064 | 17.9 | 2.878 | 0.417 | 15.3 |
| 400 | 200 | 21.911 | 38.3 | 2.461 | 3.546 | 29.9 |
| 600 | 300 | 35.911 | 56.6 | 4.044 | 15.346 | 45.8 |
| 800 | 400 | 50.654 | 74.7 | 4.767 | 48.985 | 63.3 |
| 1000 | 500 | 59.787 | 104.1 | 3.478 | 102.807 | 77.8 |
| 1200 | 600 | 69.643 | 125.3 | 3.214 | 200.231 | 92.4 |
| 1400 | 700 | 89.884 | 142.0 | 2.528 | 367.642 | 111.1 |
| 1600 | 800 | 99.701 | 160.3 | 6.456 | 578.083 | 125.4 |
| 1800 | 900 | 97.310 | 161.4 | 3.280 | 865.339 | 126.0 |
| 2000 | 1000 | 125.431 | 198.8 | 3.767 | 1662.261 | 158.6 |

**Table 6.** Results for random instances of SR, Lasso vs. matroid exchange

We observe that the objective value of the solution computed by our heuristics does not seem to increase significantly with the size of the instances, whereas the solution obtained by Lasso deteriorates quickly. Our solutions are up to 30 times better than those computed by Lasso. The one-norm of the solutions computed by our heuristic is generally much larger than the original bound $\tau$ for the Lasso approach. This gives an impression about the difference between the Lasso approach and an approach that explicitly bounds the cardinality.

We performed a similar set of experiments with the SparseNet algorithm [10], using the R package `sparsenet`. As before, we let the latter approach choose the sparsity $k$ and fix $k$ when running our algorithm. The solution of SparseNet depends on two parameters $\lambda$ and $\gamma$. The former is the degree of regularization, while the latter allows to move between the norm $||.||_1$ (as in Lasso), which is reached when $\gamma \to 0+$, and $||.||_0$, which is approximated when $\gamma \to \infty$. In our experiments, we set $\gamma = 10^6$ and choose $\lambda$ in a range that yields solutions of different cardinalities. Since SparseNet, different from our algorithm, allows for an intercept, we integrate the latter into the solution given by SparseNet, by adapting the right hand side $b$. In other words, we fix the choice of the intercept before running our algorithm.

In our experiments, we produced instances by fixing $n = 1000$ and randomly choosing $m \in \{1000, \dots, 10999\}$. All entries of $A$ and $b$ were chosen randomly, following a standard normal distribution. The parameter $\lambda$ was chosen as $10^{-5} \cdot 2^r$ for random $r \in [0, 12)$. In Figure 1, we plot the results for 775 instances. For each instance, let $\text{opt}_1$ and $\text{opt}_2$ denote the optimal values of the best found solutions by SparseNet and by our algorithm, respectively. For each instance, we added a data point at x-value $\frac{k}{n}$ and y-value $(\text{opt}_1 - \text{opt}_2)/(\text{opt}_1 + \text{opt}_2)$. In particular, all y-values are in the range $[-1, 1]$ and a large y-value means that our approach found a better solution.

It turns out that our approach never computed a solution worse than the one obtained by SparseNet. Often, the solutions are close to each other, but in some cases our algorithm clearly outperforms SparseNet, particularly for medium densities. Recall that the cardinality is not fixed for SparseNet, so that the experimental setup is actually in favor of SparseNet, while our approach has to

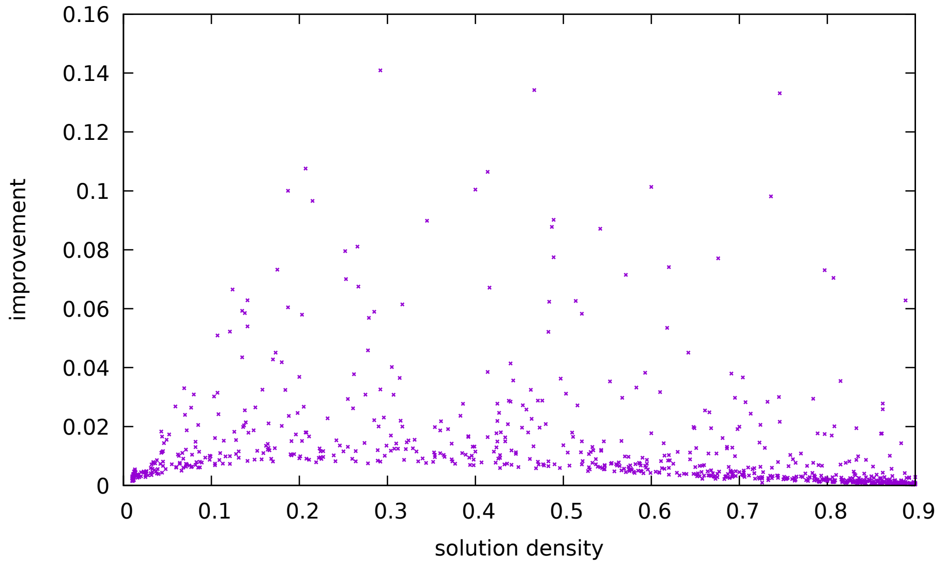


**Fig. 1.** Comparison with SparseNet

produce a solution with the same cardinality as the one obtained by SparseNet. In view of this, the results of Figure 1 show clearly that our approach is a promising method for solving sparse regression problems. We should mention however that our method takes considerably more time than SparseNet.

### 4.3 Tree underestimator problem

In [4], the authors presented an approach for computing lower bounds for quadratic binary optimization problems of the form

$$\begin{array}{ll} \min & z^\top Q z \\ \text{s.t.} & z \in \{0,1\}^n \ . \end{array} \tag{12}$$

The idea is to replace $Q$ by a diagonal matrix $X$ with $X \preccurlyeq Q$. The resulting problem is separable and can be solved by a linear optimization oracle, using the binarity of the variables; its optimal value yields a lower bound of (12). These bounds can be improved by considering more general matrices $X$: as long as the support of $X$ corresponds to a tree in the complete graph, with vertices corresponding to rows of $X$, Problem (12) can still be solved efficiently. This leads to the sparse semidefinite program

$$\begin{array}{ll} \max & \langle J_n, X\rangle \\ & Q \succcurlyeq X \\ & X_{ij} = 0 \quad \text{if } t_{ij} = 0,\ i \neq j \\ & t \in T\ , \end{array} \tag{13}$$

where $Q \in \mathbb{R}^{n\times n}$ is a symmetric matrix, $J_n \in \mathbb{R}^{n\times n}$ is the all-ones matrix, and $T$ is the set of incidence vectors of spanning forests in $K_n$. As $T$ is a matroid here, we can use the update rule based on pairwise exchanges; see Section 3.

We compare the dual bounds obtained from the new tree underestimators with the ones obtained from separable underestimators as in [4]. Our test-bed consists of the matrices $Q$ used in the binary quadratic instances of the Biq-mac library [12]. The linear part is always zero. There are 165 instances in total, subdivided into three problem classes `beasley`, `be`, and `gka`, with a number of variables ranging from 20 to 500. The results are given in Table 7. For each group of instances, we state the number of corresponding instances (`# inst`) as well as the following average information: the total cpu time (in seconds) needed by our algorithm to terminate (`time`), which is mostly spend for iteratively solving the semidefinite problem (13) for fixed $t$, the number of iterations (`# iter`), and the dual bound provided by the separable underestimator and by the tree underestimator (`sep bnd` and `tree bnd`).

| type | size | # inst | time | # iter | sep bnd | tree bnd |
|---|---|---|---|---|---|---|
| beasley | 50 | 10 | 0.60 | 19.60 | −9057.68 | −5871.09 |
| beasley | 100 | 10 | 8.18 | 44.90 | −28584.30 | −19401.50 |
| beasley | 250 | 10 | 131.23 | 93.10 | −126256.12 | −93784.32 |
| beasley | 500 | 10 | 1446.01 | 165.00 | −371126.76 | −289109.91 |
| be | 100 | 10 | 4.73 | 31.50 | −49881.47 | −34742.32 |
| be | 120 | 20 | 9.19 | 43.35 | −47460.84 | −33484.71 |
| be | 150 | 20 | 21.35 | 51.55 | −67363.73 | −49104.10 |
| be | 200 | 20 | 48.60 | 68.30 | −105519.61 | −77984.56 |
| be | 250 | 10 | 98.95 | 77.80 | −64013.84 | −47937.08 |
| gka | 20 | 1 | 0.06 | 16.00 | −5583.52 | −3742.11 |
| gka | 30 | 3 | 0.23 | 19.00 | −10168.52 | −6523.19 |
| gka | 40 | 2 | 0.21 | 11.50 | −12804.97 | −8474.36 |
| gka | 50 | 4 | 0.73 | 22.25 | −13681.36 | −9541.97 |
| gka | 60 | 3 | 0.76 | 16.67 | −17710.67 | −12311.41 |
| gka | 70 | 3 | 2.01 | 26.67 | −22796.77 | −14993.31 |
| gka | 80 | 3 | 2.81 | 32.33 | −25439.37 | −16987.01 |
| gka | 90 | 2 | 4.49 | 34.50 | −30978.50 | −22630.76 |
| gka | 100 | 13 | 5.91 | 36.54 | −34381.92 | −24873.15 |
| gka | 125 | 1 | 16.54 | 57.00 | −79859.91 | −57941.10 |
| gka | 200 | 5 | 60.70 | 81.00 | −77180.38 | −56676.81 |
| gka | 500 | 5 | 1460.60 | 162.20 | −407649.86 | −319289.17 |

**Table 7.** Lower bounds obtained by tree underestimators

The results show that the use of tree underestimators improves the obtained dual bounds significantly with respect to the separable underestimators. The running times are small enough to be clearly dominated by the time needed for the main phase of the approach presented in [4].

## 5 Conclusion

We have presented a general heuristic approach for convex optimization problems subject to combinatorial sparsity constraints. The approach is particularly successful when the notion of sparsity is defined by a matroid, so that the support can be updated by pairwise exchanges. The main idea is to use dual information to decide which variable to move to the support. Experimental results for different types of problems show that our algorithm often finds solutions being close to optimal. In particular, we show that it outperforms state-of-the-art heuristics for sparse linear regression problems in terms of solution quality.

## References

1. Bertsimas, D., Van Parys, B.: Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. arXiv preprint arXiv:1709.10029 (2017)
2. Borchers, B.: CSDP, a C library for semidefinite programming. Optimization Methods and Software **11/12**(1–4), 613–623 (1999)
3. Brodie, J., Daubechies, I., De Mol, C., Giannone, D., Loris, I.: Sparse and stable Markowitz portfolios. PNAS **106**(30), 12267–12272 (2009)
4. Buchheim, C., Traversi, E.: Quadratic combinatorial optimization using separable underestimators. INFORMS Journal on Computing **30**(3) (2018)
5. IBM ILOG CPLEX Optimizer 12.6. `www.ibm.com/software/integration/optimization/cplex-optimizer` (2015)
6. de Farias Jr., I.R., Nemhauser, G.L.: A polyhedral study of the cardinality constrained knapsack problem. Mathematical Programming **96**, 439–467 (2003)
7. DeMiguel, V., Garlappi, L., Nogales, F.J., Uppal, R.: A generalized approach to portfolio optimization: Improving performance by constraining portfolio norms. Management Science **55**(5), 798–812 (2009)
8. Dong, H., Chen, K., Linderoth, J.: Regularization vs. relaxation: A conic optimization perspective of statistical variable selection. arXiv preprint arXiv:1510.06083 (2015)
9. Kyrillidis, A., Becker, S., Cevher, V., Koch, C.: Sparse simplex projections for portfolio optimization. In: IEEE Global Conference on Signal and Information Processing (2013)
10. Mazumder, R., Friedman, J.H., Hastie, T.: Sparsenet: Coordinate descent with nonconvex penalties. Journal of the American Statistical Association **106**(495), 1125–1138 (2011)
11. Natarajan, B.K.: Sparse approximate solutions to linear systems. SIAM Journal on Computing **24**(2), 227–234 (1995)
12. Rendl, F., Rinaldi, G., Wiegele, A.: Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming **121**(2), 307 (2010)
13. van den Berg, E., Friedlander, M.: Sparse optimization with least-squares constraints. SIAM Journal on Optimization **21**(4), 1201–1229 (2011)