COMPRESSED SENSING WITH UNKNOWN SENSOR PERMUTATION

Valentin Emiya[⋆], *Antoine Bonnefoy*[⋆], *Laurent Daudet*[†], *Rémi Gribonval*[°]

[⋆] Aix-Marseille Université, CNRS UMR 7279 LIF
[°]Inria
[†] Institut Langevin, CNRS UMR 7587, Univ. Paris Diderot, ESPCI

ABSTRACT

Compressed sensing is the ability to retrieve a sparse vector from a set of linear measurements. The task gets more difficult when the sensing process is not perfectly known. We address such a problem in the case where the sensors have been permuted, *i.e.*, the order of the measurements is unknown. We propose a branch-and-bound algorithm that converges to the solution. The experimental study shows that our approach always retrieves the unknown permutation, while a simple convex relaxation strategy almost always fails. In terms of its time complexity, we show that the proposed algorithm converges quickly with respect to the combinatorial nature of the problem.

***Index Terms*—** Inverse problem; sparsity; compressed sensing; dictionary learning; permutation; optimization; branch and bound.

1. INTRODUCTION

Many studies have addressed the linear sparse estimation problem where an observed data $\mathbf{y} \in \mathbb{R}^M$ is modeled as $\mathbf{y} = \mathbf{D}\mathbf{x}$ where $\mathbf{D} \in \mathbb{R}^{M \times K}$, $M \leq K$ is the known *dictionary* or *measurement matrix* and the unknown representation $\mathbf{x} \in \mathbb{R}^K$ of the data in $\mathbf{D}$ is assumed to be sparse. Sparsity means that the number of non-zero elements in $\mathbf{x}$, denoted by $\|\mathbf{x}\|_0$, is small compared to M. A good estimation [1] of $\mathbf{x}$ from $\mathbf{y}$ and $\mathbf{D}$ is obtained by solving, for $p = 1$,

$$\arg\min_{\mathbf{x}} \|\mathbf{x}\|_p \text{ s.t. } \mathbf{y} = \mathbf{D}\mathbf{x}, \tag{1}$$

which is a convex relaxation of the NP-hard problem when $p = 0$.

In many scenarios, the dictionary $\mathbf{D}$ is unknown. It can be learned from a collection of N examples given as the columns of a matrix $\mathbf{Y} \in \mathbb{R}^{M \times N}$, each of them being sparse in the unknown dictionary. This task is known as dictionary learning [2, 3, 4] and its principle can be formulated, for $p = 0$ or $p = 1$, as the problem

$$\arg\min_{\mathbf{D},\mathbf{X}} \|\mathbf{X}\|_p \text{ s.t. } \mathbf{Y} = \mathbf{D}\mathbf{X}. \tag{2}$$

Dictionary learning is a difficult nonconvex problem. Adding prior knowledge is an option to make it simpler – yet still interesting in order to partially learn a dictionary from examples [5, 6, 7, 8]. This strategy may even lead to a convex problem, such as the estimation of gains in the measurement matrix [8].

In this paper, we address a new problem, in the same philosophy, by considering the case where the order of the sensors has been lost during the sensing process: the dictionary is known up to a permutation of its rows. Such a scenario may be of interest in applications like bit-rate reduction in a channel that does not preserve data order. It may also happen in an experimental setup with a large number of sensors – *e.g.*, a microphone array –, due to some handcrafting mistakes in the wiring between sensors and A/D converters. We call this problem the sensor permutation problem[1].

Formally, an unknown permutation matrix $\mathbf{P}$ is such that $\mathbf{PY} = \mathbf{DX}$. The related estimation problem addressed in this paper is

$$\arg\min_{\mathbf{P},\mathbf{X}} \|\mathbf{X}\|_1 \text{ s.t. } \begin{cases} \mathbf{P} \in \mathcal{P}_M \\ \mathbf{PY} = \mathbf{DX} \end{cases} \tag{3}$$

where $\mathcal{P}_M$ is the set of $M \times M$ permutation matrices. Problem (3) is combinatorial since $\mathcal{P}_M$ is discrete with size $M!$. Our contributions to overcome this complexity and solve the problem are:

- we study a convex relaxation of (3) by replacing $\mathcal{P}_M$ by its convex hull and show that in practice, this strategy almost always fails to identify the true permutation (Section 2);
- we propose a branch-and-bound algorithm that converges to the solution of (3) by exploring $\mathcal{P}_M$ cleverly (Section 3);
- we show experimentally that in many configurations, the proposed algorithm converges quickly, after exploring a small subset of $\mathcal{P}_M$; we finally check that in the experiments, the solution of (3) is indeed the original permutation (Section 4).

2. RELAXED CONVEX PROBLEM

The only reason why problem (3) is not convex is that $\mathcal{P}_M$ is not a convex set. A natural three-step strategy to obtain a tractable solution is: replace $\mathcal{P}_M$ by its convex hull; solve the resulting convex problem; project the estimate back to $\mathcal{P}_M$. The convex hull of $\mathcal{P}_M$ is the set $\mathcal{B}_M$ of $M \times M$ bistochastic matrices, *i.e.*, matrices with non-negative coefficients summing to one along each row and each column. The convex problem of interest is thus

$$\begin{aligned} \widehat{\mathbf{B}}, \widehat{\mathbf{X}} &\triangleq \arg\min_{\mathbf{B} \in \mathcal{B}_M, \mathbf{X}} \|\mathbf{X}\|_1 \text{ s.t. } \mathbf{BY} = \mathbf{DX} \\ &= \arg\min_{\mathbf{B},\mathbf{X}} \|\mathbf{X}\|_1 \text{ s.t. } \begin{cases} \mathbf{B} & \in \mathbb{R}_+^{M \times M} \\ \mathbf{B}\mathbf{1}_M & = \mathbf{1}_M \\ \mathbf{B}^T\mathbf{1}_M & = \mathbf{1}_M \\ \mathbf{BY} & = \mathbf{DX} \end{cases} \end{aligned} \tag{4}$$

where $\mathbf{1}_M$ is an M-length vector filled with ones.

This work was supported in part by Agence Nationale de la Recherche (ANR), project GRETA 12-BS02-004-01 and by the European Research Council, PLEASE project (ERC-StG-2011-277906). This project originated in Roscoff, at a workshop of the LASCARS consortium.

[1]Note that this problem is not a particular case of dictionary selection as defined in [6, 7], even if the search space is composed of a finite number of dictionaries.

Since $\mathcal{P}_M \subset \mathcal{B}_M$, the estimate $\widehat{\mathbf{B}}$ is generally not a permutation matrix : it can be post-processed by the Hungarian algorithm [9] in order to obtain the closest permutation matrix $\widetilde{\mathbf{P}} \triangleq \arg\max_{\mathbf{P}\in\mathcal{P}_M} \langle \mathbf{P}, \widehat{\mathbf{B}} \rangle$ where $\langle A, B \rangle \triangleq \sum_{i,j} A(i,j) B(i,j)$. By fixing $\widetilde{\mathbf{P}}$, the sparse representation is finally refined as

$$\widetilde{\mathbf{X}} \triangleq \arg\min_{\mathbf{X}} \|\mathbf{X}\|_1 \text{ s.t. } \widetilde{\mathbf{P}}\mathbf{Y} = \mathbf{D}\mathbf{X}. \quad (5)$$

This strategy is summarized in Algorithm 1 where any convex optimization method is suitable to solve (4) and (5).

Algorithm 1 Simple convex relaxation strategy

```
Require: $\mathbf{Y}, \mathbf{D}$.
 1: Estimate the bistochastic matrix $\widehat{\mathbf{B}}$ by solving (4).
 2: Find the closest permutation $\widetilde{\mathbf{P}}$ by applying the Hungarian algorithm to $\widehat{\mathbf{B}}$.
 3: Estimate the sparse matrix $\widetilde{\mathbf{X}}$ by solving (5).
 4: return $\widetilde{\mathbf{P}}$, $\widetilde{\mathbf{X}}$.
```

In practice, this strategy gives poor results : it almost never identifies the solution of problem (3). As an illustration, the average number of errors in the estimated permutation has been measured in a basic experimental setting, using uniformly-random permutations on M measures, a Gaussian $M \times 2M$ dictionnary $\mathbf{D}$ and $N = 40$ examples with $K_0 = 3$ non-zero Gaussian components randomly located. An error is defined as a wrong sensor index out of the M possible positions in the permutation and the number of errors is averaged among 20 trials. As reported in Table 1 for $6 \le M \le 20$, most of the permutations are far from being retrieved since the number of errors almost equals the number of elements in the permutations.

M	6	8	10	12	14	16	18	20
#errors	4.3	5.7	8.9	10.6	11.9	14.6	16.6	18.7

Table 1. Average number of errors in the permutation estimated by the simple relaxation strategy (Algorithm 1) as a function of M.

3. A BRANCH-AND-BOUND ALGORITHM

3.1. Branch-and-bound principle

Branch and bound [10, 11] is a clever way to explore a non-convex set and solve an optimization problem over this set. If the optimum is unique, it converges to the solution, with no theoretical speed guarantee – *i.e.*, the entire set is explored in the worst case. However, in many practical instances, a very significant speedup is obtained.

Let us consider the optimization problem over a discrete set $\mathcal{S}^o$

$$\boldsymbol{\mathcal{P}}(\mathcal{S}^o) : \arg\min_{x\in\mathcal{S}^o} f(x). \quad (6)$$

We denote its solution by $x^*_{\mathcal{S}^o}$ and the optimal value by $f^*_{\mathcal{S}^o} = f(x^*_{\mathcal{S}^o})$. For any subset $\mathcal{S} \subset \mathcal{S}^o$, we consider subproblem $\boldsymbol{\mathcal{P}}(\mathcal{S})$ and we assume that one can efficiently compute a lower bound $l_{\boldsymbol{\mathcal{P}}}(\mathcal{S})$ and an upper bound $u_{\boldsymbol{\mathcal{P}}}(\mathcal{S})$ on the optimal value of $\boldsymbol{\mathcal{P}}(\mathcal{S})$:

$$l_{\boldsymbol{\mathcal{P}}}(\mathcal{S}) \le f^*_{\mathcal{S}} \le u_{\boldsymbol{\mathcal{P}}}(\mathcal{S}). \quad (7)$$

By splitting the entire set $\mathcal{S}^o = \bigcup_{n=1}^N \mathcal{S}_n$ into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_N$ and defining their lowest upper bound $U \triangleq \min_n u_{\boldsymbol{\mathcal{P}}}(\mathcal{S}_n)$, we have

$$\forall n,\, U < l_{\boldsymbol{\mathcal{P}}}(\mathcal{S}_n) \Rightarrow x^*_{\mathcal{S}^o} \notin \mathcal{S}_n. \quad (8)$$

Algorithm 2 Generic branch-and-bound procedure

```
Require: A problem $\boldsymbol{\mathcal{P}}$ over a set $\mathcal{S}^o$.
 1: Initialize list of active subsets $\mathcal{A} \leftarrow \{\mathcal{S}^o\}$
 2: repeat
 3:     Choose an active subset $\mathcal{S} \in \mathcal{A}$ and remove it from $\mathcal{A}$.
 4:     Split $\mathcal{S} = \bigcup_{n=1}^N \mathcal{S}_n$ into $\mathcal{S}_1, \ldots, \mathcal{S}_N$.
 5:     Add new active subsets $\mathcal{S}_1, \ldots, \mathcal{S}_N$ in $\mathcal{A}$.
 6:     Compute lower bounds $l_{\boldsymbol{\mathcal{P}}}(\mathcal{S}_1), \ldots, l_{\boldsymbol{\mathcal{P}}}(\mathcal{S}_N)$ and upper bounds $u_{\boldsymbol{\mathcal{P}}}(\mathcal{S}_1), \ldots, u_{\boldsymbol{\mathcal{P}}}(\mathcal{S}_N)$.
 7:     Update $L$ (resp. $U$) as the lowest lower bound (resp. lowest upper bound) among all the elements of $\mathcal{A}$.
 8:     Prune every subset $\mathcal{S} \in \mathcal{A}$ such that $l_{\boldsymbol{\mathcal{P}}}(\mathcal{S}) > U$.
 9: until $L = U$, i.e., $\mathcal{A}$ is reduced a singleton
10: return the solution $x$ remaining in $\mathcal{A} = \{\{x\}\}$.
```

Hence, the knowledge of lower and upper bounds can help to discard some subsets without exploring them exhaustively. A branch-and-bound strategy uses these ingredients to create subsets (branching) and bound their optimal values in a iterative way until all the space but one element – the solution – has been pruned. The procedure is summarized in Algorithm 2. Starting from the full set $\mathcal{S}^o$, a list of active subsets is iteratively built. At each iteration, an active subset may be split into new active subsets, and some active subsets may be pruned using (8), avoiding an exhaustive evaluation of the problem. In practice, designing such a procedure relies on the ability to:

- compute a lower bound $l_{\boldsymbol{\mathcal{P}}}(\mathcal{S})$, for instance by relaxing $\mathcal{S}$ to a convex set $\mathcal{S}'$ and by solving $\boldsymbol{\mathcal{P}}(\mathcal{S}')$;
- compute an upper bound $u_{\boldsymbol{\mathcal{P}}}(\mathcal{S})$, for instance by projecting the solution of the relaxed problem back onto $\mathcal{S}$ to obtain a suboptimal feasible solution;
- exhibit a heuristic to choose which active subset to split (line 3 in Algorithm 2): a common heuristic is to select the active subset with lowest lower bound.
- exhibit a heuristic to split the selected subset (line 4 in Algorithm 2), usually resulting into subsets with balanced sizes.

3.2. Proposed algorithm

We propose a strategy to explore the set of permutations $\mathcal{P}_M$ by constraining some elements to equal one in the permutation matrices. An active subset of $\mathcal{P}_M$ is fully characterized by such a constraint set. Fig. 1 illustrates the tree structure underlying the exploration of $\mathcal{P}_3$. Each active subset of $\mathcal{P}_M$ is split by choosing a row in which there is no constrained element and by creating a subset with such a constraint for each possible position in this row. The proposed procedure is given in Algorithm 3 and its key steps are detailed below.
Problems over subsets. An active subset of $\mathcal{P}_M$ is fully characterized by a set of positions $\mathcal{C} \subset \{1, \ldots, M\}^2$ to constrain the related elements of $\mathbf{P}$ to equal one. Thus, the restriction of problem (3) to any subset of $\mathcal{P}_M$ characterized by a constraint set $\mathcal{C}$ is defined as

$$\boldsymbol{\mathcal{P}}_{\mathbf{Y},\mathbf{D}}(\mathcal{C}) : \arg\min_{\mathbf{P},\mathbf{X}} \|\mathbf{X}\|_1 \text{ s.t. } \begin{cases} \mathbf{P} \in \mathcal{P}_M \\ \mathbf{P}(i,j) = 1, \forall (i,j) \in \mathcal{C} \\ \mathbf{P}\mathbf{Y} = \mathbf{D}\mathbf{X} \end{cases} \quad (9)$$

Note that the first two constraints in (9) are equivalent to fixing the lines and columns of $\mathbf{P}$ that contain one constraint from $\mathcal{C}$ and finding an unconstrained permutation matrix in $\mathcal{P}_{M-|\mathcal{C}|}$.

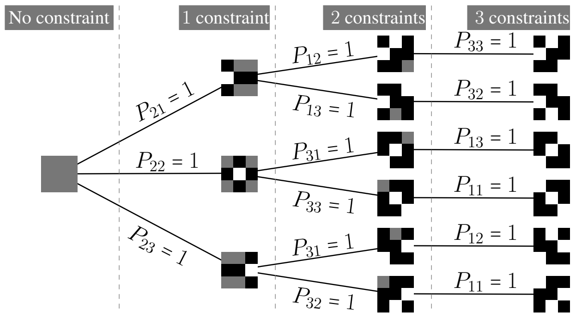

Fig. 1. Tree structure to explore $\mathcal{P}_3$ by adding constraints (white ones) on the permutation matrix. Some elements are consequently zeros (black), others are free (gray).

Algorithm 3 Proposed branch-and-bound algorithm

```
Require: $\mathbf{Y}, \mathbf{D}$.
1: Initialize list for active constraint sets $\mathcal{A} \leftarrow \{\{\varnothing\}\}$
2: repeat
3:     Select $\mathcal{C} \in \mathcal{A}$ with lowest lower bound, remove it from $\mathcal{A}$.
4:     Choose splitting point $(i_0, j_0)$ as the location of the greatest
       element in the bistochastic matrix obtained from $l_{\mathbf{Y},\mathbf{D}}(\mathcal{C})$.
5:     Add $\mathcal{C} \cup \{(i_0, j)\}$ to $\mathcal{A}$ for each $j$ such that
       $\mathcal{P}_{\mathbf{Y},\mathbf{D}}(\mathcal{C} \cup \{(i_0, j)\})$ is feasible.
6:     Compute the lower bound $l_{\mathbf{Y},\mathbf{D}}(\mathcal{C} \cup \{(i_0, j)\})$ and the upper
       bound $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C} \cup \{(i_0, j)\})$ of each added constraint set.
7:     Update $L$ (resp. $U$) as the lowest lower bound (resp. lowest
       upper bound) among all the elements of $\mathcal{A}$.
8:     Prune every $\mathcal{S} \in \mathcal{A}$ such that $l_{\mathcal{P}}(\mathcal{S}) > U$.
9: until $L = U$
10: return permutation matrix $\widehat{\mathbf{P}}$ and sparse matrix $\widehat{\mathbf{X}}$ related to the
    only element induced by the unique constraint set in $\mathcal{A} = \{\mathcal{C}\}$.
```

Lower and upper bounds. By relaxing $\mathcal{P}_M$ to its convex hull, we define the following tractable lower bound on the solution over any subset constrained by $\mathcal{C}$:

$$l_{\mathbf{Y},\mathbf{D}}(\mathcal{C}) : \min_{\mathbf{B},\mathbf{X}} \|\mathbf{X}\|_1 \text{ s.t. } \begin{cases} \mathbf{B} \in \mathbb{R}_+^{M \times M} \\ \mathbf{B}\mathbf{1}_M = \mathbf{1}_M \\ \mathbf{B}^T\mathbf{1}_M = \mathbf{1}_M \\ \mathbf{B}(i,j) = 1, \forall (i,j) \in \mathcal{C} \\ \mathbf{B}\mathbf{Y} = \mathbf{D}\mathbf{X} \end{cases} \tag{10}$$

The upper bound $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C})$ is computed from the optimal bistochastic matrix obtained in (10) as follows : the closest permutation matrix is computed by the Hungarian algorithm applied in $\mathcal{P}_{M-|\mathcal{C}|}$ after removing all contrained lines and columns; the resulting permutation matrix $\widetilde{\mathbf{P}} \in \mathcal{P}_M$ is used to solve (5); the upper bound is defined as the l_1 norm $\|\widetilde{\mathbf{X}}\|_1$ of the solution. It may happen that if $\mathcal{C}$ is a set of contraints obtained from a parent set $\mathcal{C}'$, $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C})$ may be greater than $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C}')$ while the solution for $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C}')$ does not violate constraints $\mathcal{C}$. In this case, $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C})$ is set to $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C}')$.

Branching : choosing a new constraint. In order to build active subsets, a natural heuristic consists in choosing the maximum unconstrained element indexed by $(i_0, j_0) \notin \mathcal{C}$ in the bistochastic matrix estimated in (10). Indeed, it is likely that contraining this element to equal one will be a step toward the optimal solution, and that alternative subsets in which this element equals zero will be pruned early. Based on this heuristic, we explore constrained sets where the constraint is moved along line i_0[2] of the permutation matrix (line 5 in Algorithm 3). One must ensure that the problem is feasible on the new set $\mathcal{C} \cup \{(i_0, j)\}$ by checking that $(i', j), (i_0, j') \notin \mathcal{C}, \forall i', j'$, *i.e.*, the element (i_0, j) in the permutation matrix is not induced to equal zero by an existing constraint in $\mathcal{C}$.

4. EXPERIMENTS

The experiments rely on synthetic data generated randomly as follows : the $M \times K$ dictionary is composed of standard Gaussian i.i.d. entries; matrix $\mathbf{X}$ is composed by N K_0-sparse columns, with standard Gaussian i.i.d. non-zero entries; a permutation matrix is drawn uniformly from $\mathcal{P}_M$; the noiseless observation matrix is computed as $\mathbf{Y} = \mathbf{P}^T\mathbf{D}\mathbf{X}$. Dimensions M, K, K_0, N are detailed for each experiment.

The code is written in Matlab and uses the CVX toolbox [12] to solve the relaxed problems (10), an existing implementation of the Hungarian algorithm [13], and the L1 magic toolbox [14] to solve (5). The code has been run on a 2.9GHz core. All the code and data are provided with this paper for reproducible research purposes[3].

4.1. Typical behavior

In order to illustrate how the algorithm behaves along iterations, a representative example is shown in Fig. 2 with $M = 10$, $K = 20$, $K_0 = 3$ and $N = 100$. The algorithm converges to the true solution in 265 iterations, after exploring 1671 subproblems, *i.e.*, about 0.05% of the size $M! = 10! \approx 3.6 \cdot 10^6$ of $\mathcal{P}_M$. The list of active constraint sets grows in the first 10 iterations, the active sets $\mathcal{C}$ having 1 to 3 constraints. The lowest upper bound U in blue shows that the best candidate encountered is refined along the iterations and that the first one is not optimal. In the next 100 iterations, the exploration of the active constraint sets results in pruning most of the active sets obtained after splitting, their lower bound being greater than the lowest upper bound U. As a result, the list tends to shrink and the number of constraints does not exceed 4. At the very end, moving from 4 to 10 constraints, *i.e.*, to the solution, is performed in very few iterations.

This example shows the benefits of the proposed approach: starting from the non-optimal permutation given by Algorithm 1, it explores the low-depth nodes of the tree (*i.e.*, active sets with few constraints, see Fig. 2 bottom) which happens to be sufficient to discard most of the search space without exploring it and to converge to the true solution.

4.2. Global trends

Extensive experiments are conducted in order to depict the performance of the proposed algorithm, in terms of time complexity – *i.e.*, the number of subproblems explored before converging, averaged among several trials – and of recovery of the original permutation.

Time complexity vs. sparsity and redundancy. Time complexity has been measured when fixing $M = 8$, $N = 60$ and changing the sparsity K_0 and the redundancy M/K. Average results over 10 trials are shown in Fig. 3. Convergence is typically achieved in less than 100 iterations for very sparse signals and slightly-redundant dictionaries – the set of permutations having $8! \approx 40320$ elements. Time complexity increases moderately with K_0, more strongly with K.

Time complexity vs. permutation size. The increase of the time complexity with the permutation size M is a key question since the

[2]Moving the constraint along column j_0 is likely to give similar results.

[3]http://www.lif.univ-mrs.fr/ vemiya/icassp2014/

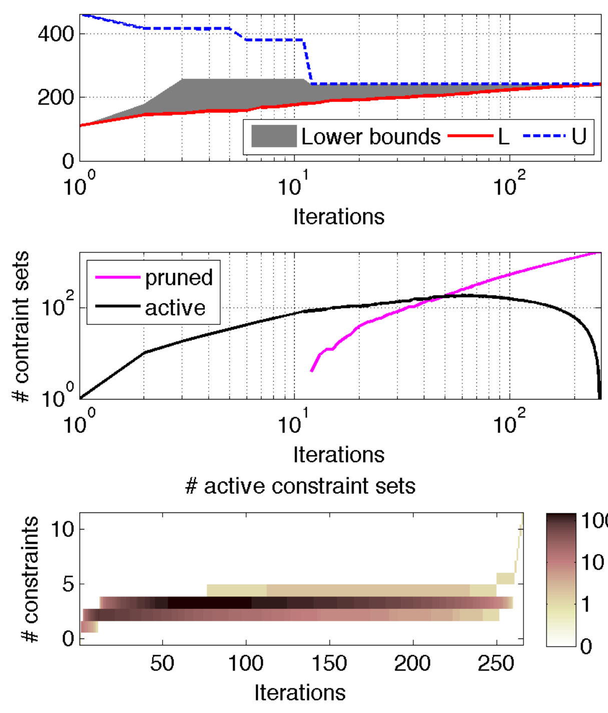

Fig. 2. Typical behavior of the proposed algorithm along the iterations. Top: lowest lower bound L, lowest upper bound U and area representing the range of all active lower bounds in $\mathcal{A}$. Middle: number of active and pruned constraint sets in $\mathcal{A}$. Bottom: histogram of the number of constraints among elements in $\mathcal{A}$.

size of $\mathcal{P}_M$ grows as $M! \sim \sqrt{2\pi M}\left(\frac{M}{e}\right)^M$ and since the branch-and-bound procedure does not provide any theoretical convergence rate. Results are reported in Fig. 4 for $M \in [6, 14]$, $K = 20$, $K_0 = 3$ and $N = 60$. The time complexity increases from 90 to 17000 but the number of explored subproblems gets dramatically low compared to the size of $\mathcal{P}_M$ ($2 \cdot 10^{-5}\%$ for $M = 14$). This result supports the idea that solving combinatorial problem (3) is not hopeless.

Time complexity vs. number of examples. Time complexity has been measured when fixing $M = 10$, $K = 20$, $K_0 = 3$, and changing the number of examples N. Median results and quartiles are represented in Fig. 5. As expected, the convergence needs to explore less subproblems when more examples are available. However, each subproblem demands more time, due to the size of the data, so that a tradeoff may be achieved by adjusting N. The quartiles represented in Fig. 5 also illustrate the distribution of actual time complexities which may deviate significantly from average or median results.

Perfect permutation recovery. Finally, one may wonder whether the estimated permutation equals the original permutation or whether some error rates must be measured. Actually, the true original permutation is retrieved by our algorithm in *all* the experiments above.

5. CONCLUSIONS

We have defined the sensor permutation problem in compressed sensing and proposed a branch-and-bound algorithm to solve it. A number of experiments have demonstrated that sparsity is a good regularization to recover an unknown permutation: the proposed method based on non-convex optimization is tractable and gives perfect recovery results, while convex optimization fails in retrieving the original permutation.

This work opens a number of new directions. One may generalize the sensor permutation problem to a larger class of objective functions, including noise in the data fitting term or structured sparsity. Other discrete optimization problems in compressed sensing may be defined and addressed using branch-and-bound strategies: instead of a permutation matrix, one can consider an unknown diagonal matrix with diagonal elements in $\{-1, 1\}$ to model unknown sensor polarization, or in $\{0, 1\}$ to model unknown defective sensors. Time complexity of such brand-and-bound algorithms may be improved by investigating other heuristics and by adjusting the number of examples to find a balance between the time needed to exploit each subproblem and the time spent on exploring more subproblems. Early stopping may also be used to control the time complexity at the price of losing the perfect recovery guarantee.

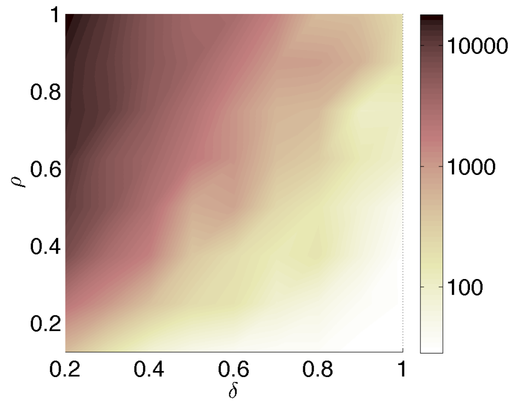

Fig. 3. Average number of explored subproblems as a function of $\delta = \frac{M}{K}$ and $\rho = \frac{K_0}{M}$, on a logarithmic color scale.

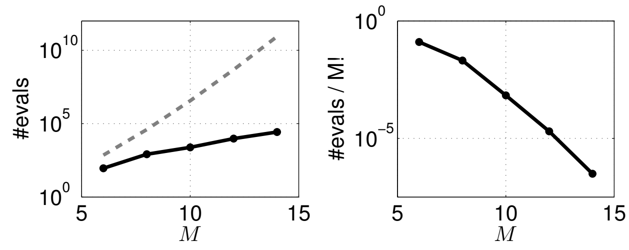

Fig. 4. Influence of the number of measures M. Left: average number of evaluated subproblems (plain) and size $M!$ of the set of permutations (dashed). Right: ratio between both quantities.

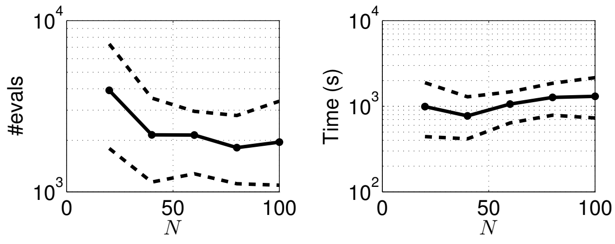

Fig. 5. Influence of the number of examples N: median (plain) and quartile (dashed) curves for the number of evaluated subproblems (left) and the computational time (right).

6. REFERENCES

[1] S. Chen, D. Donoho, and M. Saunders, “Atomic decomposition by basis pursuit,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1998.

[2] M. Aharon, M. Elad, and A. Bruckstein, “K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation,” *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[3] M. Yaghoobi, T. Blumensath, and M.E. Davies, “Dictionary learning for sparse approximations with the majorization method,” *IEEE Trans. Signal Processing*, vol. 57, no. 6, pp. 2178–2191, June 2009.

[4] R. Rubinstein, A.M. Bruckstein, and M. Elad, “Dictionaries for sparse representation modeling,” *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1045–1057, June 2010.

[5] R. Rubinstein, M. Zibulevsky, and M. Elad, “Double sparsity: Learning sparse dictionaries for sparse signal approximation,” *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1553–1564, Mar. 2010.

[6] V. Cevher and A. Krause, “Greedy dictionary selection for sparse representation,” *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 5, pp. 979–988, Sept. 2011.

[7] M. Yaghoobi, L. Daudet, and Davies. M. E., “Optimal dictionary selection using an overcomplete joint sparsity model,” *CoRR*, vol. abs/1212.2834, 2012.

[8] C. Bilen, G. Puy, R. Gribonval, and L. Daudet, “Convex Optimization Approaches for Blind Sensor Calibration using Sparsity,” *ArXiv e-prints*, Aug. 2013.

[9] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[10] E. L. Lawler and D. E. Wood, “Branch-and-bound methods: A survey,” *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[11] S. Boyd and J. Mattingley, “Branch and bound methods,” Notes for EE364b, Stanford University, Mar. 2007.

[12] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming, version 2.0 beta,” http://cvxr.com/cvx, Sept. 2013.

[13] N. Börlin, “Functions related to the assignment problem, version 1.0,” http://www8.cs.umu.se/ niclas/matlab/assignprob/, May 1999.

[14] E. Candès and J. Romberg, “l1-magic: Recovery of Sparse Signals via Convex Programming,” http://users.ece.gatech.edu/ justin/l1magic/, Oct. 2005.