

Article

Bivariate Partial Information Decomposition: The Optimization Perspective

Abdullah Makkeh * , Dirk Oliver Theis and Raul Vicente

Institute of Computer Science, University of Tartu, 51014 Tartu, Estonia; dotheis@ut.ee (D.O.T.); raul.vicente.zafra@ut.ee (R.V.)

* Correspondence: makkeh@ut.ee

Received: 7 July 2017; Accepted: 28 September 2017; Published: 7 October 2017

Abstract: Bertschinger, Rauh, Olbrich, Jost, and Ay (Entropy, 2014) have proposed a definition of a decomposition of the mutual information $MI(\mathbf{X} : \mathbf{Y}, \mathbf{Z})$ into shared, synergistic, and unique information by way of solving a convex optimization problem. In this paper, we discuss the solution of their Convex Program from theoretical and practical points of view.

Keywords: partial information decomposition; bivariate information decomposition; applications of convex optimization

1. Introduction

Bertschinger, Rauh, Olbrich, Jost, and Ay [1] have proposed to compute a decomposition of the mutual information $MI(\mathbf{X} : \mathbf{Y}, \mathbf{Z})$ into shared, synergistic, and unique contributions by way of solving a convex optimization problem. It is important to mention that William and Beer in [2] were the first to propose a measure for information decomposition. That measure suffered from serious flaws, which prompted a series of other papers [3–5] trying to improve these results. Denote by X the range of the random variable \mathbf{X} , by Y the range of \mathbf{Y} , and by Z the range of \mathbf{Z} . Further, let

$$\begin{aligned} b_{x,y}^y &= \mathbb{P}(\mathbf{X} = x \wedge \mathbf{Y} = y) && \text{for all } x \in X, y \in Y \\ b_{x,z}^z &= \mathbb{P}(\mathbf{X} = x \wedge \mathbf{Z} = z) && \text{for all } x \in X, z \in Z \end{aligned}$$

the marginals. Then the Bertschinger et al. definition of synergistic information is

$$\widetilde{CI}(\mathbf{X} : \mathbf{Y}; \mathbf{Z}) := MI(\mathbf{X} : \mathbf{Y}, \mathbf{Z}) - \min_q MI_{(x,y,z) \sim q}(x : y, z),$$

where MI denotes mutual information, and $(x, y, z) \sim q$ stands for (x, y, z) is distributed according to the probability distribution q . The minimum ranges over all probability distributions $q \in [0, 1]^{X \times Y \times Z}$ which satisfy the marginal equations

$$\begin{aligned} q_{x,y,*} &= b_{x,y}^y && \text{for all } (x, y) \in X \times Y \\ q_{x,*,z} &= b_{x,z}^z && \text{for all } (x, z) \in X \times Z. \end{aligned}$$

We use the notational convention that an asterisk “*” is to be read as “sum over everything that can be plugged in here”; e.g.,

$$q_{x,y,*} := \sum_w q_{x,y,w}.$$

(We don’t use the symbol * in any other context).

As pointed out in [1], the function $q \mapsto MI_{(x,y,z) \sim q}(x : y, z)$ is convex and smooth in the interior of the convex region, and hence, by textbook results on convex optimization, for every $\varepsilon > 0$,

an ε -approximation of the optimum—i.e., a probability distribution q^\star for which $\text{MI}_{(x,y,z)\sim q^\star}(x : y, z)$ is at most ε larger than the minimum—can be found in at most $O(\alpha^2/\varepsilon^2)$ rounds of an iterative algorithm (e.g., Proposition 7.3.1 in [6]), with a parameter α (the asphericity of the feasible region) depending on the marginals b^y, b^z . Each iteration involves evaluating the gradient of $\text{MI}_{(x,y,z)\sim q^\star}(x : y, z)$ with respect to q , and $O(|X \times Y \times Z|)$ additional arithmetic operations.

In Appendix A of their paper, Bertschinger et al. discuss practical issues related to the solution of the convex optimization problem: They analyze the feasible region, solve by hand some of the problems, and complain that Mathematica could not solve the optimization problem out of the box.

Our paper is an expansion of that appendix in [1]. Driven by the need in application areas (e.g., [7]) to have a robust, fast, out-of-the-box method for computing $\widetilde{\text{CI}}$, we review the required convex optimization background; discuss different approaches to computing $\widetilde{\text{CI}}$ and the theoretical reasons contributing to the poor performance of some of them; and finally compare several of these approaches on practical problem instance. In conclusion, we offer two different practical ways of computing $\widetilde{\text{CI}}$.

The convex optimization problem in [1] is the following:

$$\begin{aligned} & \text{minimize } \text{MI}_{(x,y,z)\sim q}(x : y, z) && \text{over } q \in \mathbb{R}^{X \times Y \times Z} \\ & \text{subject to } q_{x,y,*} = b_{x,y}^y && \text{for all } (x, y) \in X \times Y \\ & && q_{x,*,z} = b_{x,z}^z && \text{for all } (x, z) \in X \times Z \\ & && q_{x,y,z} \geq 0 && \text{for all } (x, y, z) \in X \times Y \times Z \end{aligned} \tag{CP}$$

This is a Convex Program with a non-linear, convex continuous objective function $q \mapsto \text{MI}_{(x,y,z)\sim q}(x : y, z)$, which is smooth in all points q with no zero entry. The feasible region (i.e., the set of all q satisfying the constraints) is a compact convex polyhedron, which is nonempty, as the distribution of the original random variables $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ is a feasible solution. Clearly, the condition that q should be a probability distribution is implied by it satisfying the marginal equations.

In the next section, we review some background about convex functions and convex optimization, in particular with respect to non-smooth functions. Section 3 aims to shed light on the theoretical properties of the convex program (CP). In Section 4, we present the computer experiments which we conducted and their results. Some result tables are in the Appendix A.

2. Convex Optimization Basics

Since the target audience of this paper is not the optimization community, for the sake of easy reference, we briefly review the relevant definitions and facts, tailored for our problem: the feasible region is polyhedral, the objective function is (convex and continuous but) not everywhere smooth.

Let f be a convex function defined on a convex set $C \subseteq \mathbb{R}^n$; let $x \in C$ and $d \in \mathbb{R}^n$. The *directional derivative* of f at x in the direction of d is defined as the limit

$$f'(x; d) := \lim_{\varepsilon \searrow 0} \frac{f(x + \varepsilon d) - f(x)}{\varepsilon} \in [-\infty, \infty]. \tag{1}$$

Remark 1 ([8], Lemma 2.71). *The limit in (1) exists in $[-\infty, \infty]$, by the convexity of f .*

A *subgradient* of f at a point $x \in C$ is a vector $g \in \mathbb{R}^n$ such that, for all $y \in C$, we have $f(y) \geq f(x) + g^\top(y - x)$.

There can be many subgradients of f at a point. However:

Remark 2. *If f is differentiable at x , then $\nabla f(x)$ is the only subgradient of f at x .*

Lemma 1 ([8], Lemma 2.73). *If g is a subgradient of f at $x \in C$, and $y \in C$, then $g^\top(y - x) \leq f'(x; y - x)$.*

We state the following lemma for convenience (it will be used in the proof of Proposition 2). It is a simplification of the Moreau-Rockafellar Theorem ([8], Theorem 2.85).

Lemma 2. Let $C_i \subset \mathbb{R}^\ell$, $i = 1, \dots, k$ be a closed convex sets, $f_i: C_i \rightarrow \mathbb{R}$ continuous convex functions, and

$$f(x) = \sum_{i=1}^k f_i(x_i)$$

for $x = (x_1, \dots, x_k) \in \prod_{i=1}^k C_i \subset \mathbb{R}^{(\ell)k}$. If, for $i = 1, \dots, k$, g_i is a subgradient of f_i at x_i , then $g := (g_1, \dots, g_k)$ is a subgradient of f at x . Moreover, all subgradients of f at x are of this form.

For the remainder of this section, we reduce the consideration to convex sets C of the form given in the optimization problem (CP). Let A be an $m \times n$ matrix, and b an m -vector. Assume that $C = \{x \in \mathbb{R}^n \mid Ax = b \wedge x \geq 0\}$. Suppose f is a convex function defined on C . Consider the convex optimization problem $\min_{x \in C} f(x)$.

Let $x \in C$. A vector $d \in \mathbb{R}^n$ is called a *feasible descent direction* of f at x , if

- (a) for some $\varepsilon > 0$, $x + \varepsilon d \in C$ (“feasible direction”); and
- (b) $f'(x; d) < 0$ (“descent direction”).

Clearly, if a feasible descent direction of f at x exists, then x is not a minimum of f over C . The following theorem is a direct consequence of ([8], Theorem 3.34) adapted to our situation.

Theorem 1 (Karush-Kuhn-Tucker). Suppose that for every $j = 1, \dots, n$, there is an $x \in C$ with $x_j > 0$. The function f attains a minimum over C at a point $x \in C$ if, and only if, there exist

- a subgradient g of f at x
- and an m -vector $\lambda \in \mathbb{R}^m$

such that $A^\top \lambda \leq g$, and for all $j \in \{1, \dots, n\}$ with $x_j > 0$, equality holds: $(A^\top \lambda)_j = g_j$.

The condition “ $x_j = 0$ or $(A^\top \lambda)_j = g_j$ ” is called *complementarity*.

2.1. Convex Optimization Through Interior Point Methods

For reasons which will become clear in Section 3, it is reasonable to try to solve (CP) using a so-called *Interior Point* or *Barrier Method* (we gloss over the (subtle) distinction between IPMs and BMs). The basic idea of these iterative methods is that all iterates are in the “interior” of something. We sketch what goes on.

From a *theoretical* point of view, a closed convex set C is given in terms of a *barrier function*, i.e., a convex function $F: C \rightarrow \mathbb{R}$ with $F(x) \rightarrow \infty$ whenever x approaches the boundary of C . The prototypical example is $C := [0, \infty]^n$, and $F(x) := -\sum_j \ln x_j$. The goal is to minimize a *linear* objective function $x \mapsto \sum_j c_j x_j$ over C .

To fit problems with nonlinear (convex) objective f into this paradigm, the “epigraph form” is used, which means that a new variable s is added along with a constraint $f(q) \leq s$, and the objective is “minimize s ”.

The algorithm maintains a *barrier parameter* $t > 0$, which is increased gradually. In iteration k , it solves the unconstrained optimization problem

$$x^{(k)} := \operatorname{argmin}_{x \in \mathbb{R}^n} (t_k \cdot c^\top x + F(x)) \quad (2)$$

The barrier parameter is then increased, and the algorithm proceeds with the next iteration. The fact that the barrier function tends to infinity towards the boundary of C makes sure that the iterates $x^{(k)}$ stay in the interior of C for all k .

The unconstrained optimization problem (2) is solved using Newton’s method, which is itself iterative. If x is the current iterate, Newton’s method finds the minimum to the second order Taylor expansion about x of the objective function $g: x \mapsto t_k \cdot c^\top x + F(x)$:

$$\begin{aligned} & \operatorname{argmin}_{y \in \mathbb{R}^n} \left(g(x) + \nabla g(x)^\top (y - x) + \frac{1}{2} (y - x) Hg(x) (y - x) \right) \\ & = x - Hg(x)^{-1} \nabla g(x). \end{aligned}$$

where $Hg(x)$ denotes the Hessian of g at x . Then Newton’s method updates x (e.g., by simply replacing it with the argmin).

Note that $Hg(x) = HF(x)$, so the convergence properties (as well as, whether it works at all or not), depend on the properties of the barrier function. Suitable barrier functions are known to exist for all compact convex sets [9,10]. However, for a given set, finding one which can be quickly evaluated (along with the gradient and the Hessian) is sometimes a challenge.

A more “concrete” point of view is the following. Consider f and C as above: $C := \{x \mid Ax = b, x \geq 0\}$ and f is convex and continuous on C , and smooth in the interior of C (here: the points $x \in C$ with $x_j > 0$ for all j) which we assume to be non-empty. A simple barrier-type algorithm is the following. In iteration k , solve the equality-constrained optimization problem

$$x^{(k)} := \operatorname{argmin}_{\substack{x \in \mathbb{R}^n \\ Ax=b}} \left(t_k \cdot f(x) - \sum_{j=1}^n \ln(x_j) \right). \tag{3}$$

The equality-constrained problem is solved by a variant of Newton’s method: If x is the current iterate, Newton’s method finds the minimum to the second order Taylor expansion about x of the function $g: x \mapsto t_k f(x) - \sum \ln(x_j)$ subject to the equations $Ax = b$, using Lagrange multipliers, i.e., it solves the linear system

$$\begin{aligned} Hg(x)y + A^\top \lambda &= -\nabla g(x) \\ Ax &= b, \end{aligned}$$

with $Hg(x) = t_k Hf(x) + \operatorname{Diag}_j(1/x_j^2)$ and $\nabla g(x) = t_k \nabla f(x) - \operatorname{Diag}_j(1/x_j)$, where $\operatorname{Diag}(\cdot)$ denotes a diagonal matrix of appropriate size with the given diagonal.

By convexity of f , $Hf(x)$ is positive semidefinite, so that adding the diagonal matrix results in a (strictly) positive definite matrix. Hence, the system of linear equations always has a solution.

The convergence properties of this simple barrier method now depend on properties of f . We refer the interested reader to ([11], Chapter 11) for the details.

Generally speaking, today’s practical Interior Point Methods are “Primal-Dual Interior Point Methods”: They solve the “primal” optimization problem and the dual—the system in Theorem 1—simultaneously. After (successful) termination, they return not only an ε -approximation x of the optimum, but also the Lagrange multipliers λ which certify optimality.

3. Theoretical View on Bertschinger et al.’s Convex Program

We will use the following running example in this section.

Example 1 (Part 1/4). Let $X := \{0, 1, 2, 3\}$, $Y, Z := \{0, 1\}$.

$$\begin{array}{cccc} b_{0,0}^y := 1/4 & b_{0,1}^y := 0 & b_{0,0}^z := 1/4 & b_{0,1}^z := 0 \\ b_{1,0}^y := 0 & b_{1,1}^y := 1/4 & b_{1,0}^z := 0 & b_{1,1}^z := 1/4 \\ b_{2,0}^y := 1/8 & b_{2,1}^y := 1/8 & b_{2,0}^z := 1/8 & b_{2,1}^z := 1/8 \\ b_{3,0}^y := 1/8 & b_{3,1}^y := 1/8 & b_{3,0}^z := 1/8 & b_{3,1}^z := 1/8 \end{array}$$

Among all $q \in \mathbb{R}^{X \times Y \times Z}$ (16 dimensions) satisfying the 16 equations

$$q_{x,y,*} = b_{x,y}^y \quad \forall x, y, \quad q_{x,*,z} = b_{x,z}^z \quad \forall x, z,$$

and the 16 inequalities $q_{x,y,z} \geq 0$ for all x, y, z , we want to find one which minimizes $\text{MI}_{(x,y,z) \sim q}(x : y, z)$.

3.1. The Feasible Region

In this section, we discuss the feasible region:

$$\mathbb{Q}(b) := \left\{ q \in \mathbb{R}_+^{X \times Y \times Z} \mid \forall x, y, z: q_{x,y,*} = b_{x,y}^y \ \& \ q_{x,*,z} = b_{x,z}^z \right\}$$

We will omit the “(b)” when no confusion can arise. We will always make the following assumptions:

1. b^y and b^z are probability distributions.
2. $\mathbb{Q}(b)$ is non-empty. This is equivalent to $b_{x,*}^y = b_{x,*}^z$ for all $x \in X$.
3. No element of X is “redundant”, i.e., for every $x \in X$ we have both $b_{x,*}^y > 0$ and $b_{x,*}^z > 0$.

First of all, recall the vectors $\bar{d} \in \mathbb{R}^{X \times Y \times Z}$ from Appendix A.1 of [1], defined by

$$\bar{d}^{x,y,z,y',z'} := 1^{x,y,z} + 1^{x,y',z'} - 1^{x,y,z'} - 1^{x,y',z}, \tag{4}$$

(where 1^{\dots} is the vector with exactly one non-zero entry in the given position, and 0 otherwise) satisfy $\bar{d}_{x,y,*} = \bar{d}_{x,*,z} = 0$ for all x, y, z (we omit the superscripts for convenience, when possible).

Our first proposition identifies the triples (x, y, z) for which the equation $q_{x,y,z} = 0$ holds for all $q \in \mathbb{Q}(b)$.

Proposition 1. *If $q_{x,y,z} = 0$ holds for all $q \in \mathbb{Q}(b)$, then $b_{x,y}^y = 0$ or $b_{x,z}^z = 0$.*

Proof. Let $x, y, z \in X \times Y \times Z$, and assume that $b_{x,y}^y \neq 0$ and $b_{x,z}^z \neq 0$. Take any $p \in \mathbb{Q}(b)$ with $p_{x,y,z} = 0$ —if no such p exists, we are done. Since the marginals are not zero, there exist $y' \in Y \setminus \{y\}$ and $z' \in Z \setminus \{z\}$ with $p_{x,y',z} > 0$ and $p_{x,y,z'} > 0$. Let $\bar{d} := \bar{d}^{x,y,z,y',z'}$. By the remarks preceding the proposition, $p + \delta \bar{d}$ satisfies the marginal equations for all $\delta \in \mathbb{R}$. Since $p_{x,y',z} > 0$ and $p_{x,y,z'} > 0$, there exists a $\delta > 0$ such that $q := p + \delta \bar{d} \geq 0$. Hence, we have $q \in \mathbb{Q}(b)$ and $q_{x,y,z} > 0$, proving that the equation $q_{x,y,z} = 0$ is not satisfied by all elements of $\mathbb{Q}(b)$. \square

The proposition allows us to restrict the set of variables of the Convex Program to those triples (x, y, z) for which a feasible q exists with $q_{x,y,z} \neq 0$ (thereby avoiding unnecessary complications in the computation of the objective function and derivatives; see the next section); the equations with RHS 0 become superfluous. We let

$$\mathcal{J}(b) := \left\{ (x, y, z) \in X \times Y \times Z \mid b_{x,y}^y > 0 \ \wedge \ b_{x,z}^z > 0 \right\} \tag{5}$$

denote the set of remaining triplets. We will omit the “(b)” when no confusion can arise.

Example 2 (Part 2/4). *Continuing Example 1, we see that the equations with RHS 0 allow to omit the variables $q_{x,y,z}$ for the following triples (x, y, z) :*

$$\begin{aligned} (0, 1, 0) & \quad (0, 0, 1) \\ (0, 1, 1) & \quad (0, 1, 1) \\ (1, 0, 0) & \quad (1, 0, 0) \\ (1, 0, 1) & \quad (1, 1, 0) \end{aligned}$$

Hence, $\mathcal{J} := \{(0, 0, 0), (1, 1, 1), (2, y, z), (3, y', z') \mid y, z, y', z' \in \{0, 1\}\}$ (10 variables), and we are left with the following 12 equations:

$$\begin{array}{lll}
 q_{0,0,*} = b_{0,0}^y = 1/4 & & q_{0,*0} = b_{0,0}^z = 1/4 \\
 q_{1,1,*} = b_{1,1}^y = 1/4 & & q_{1,*1} = b_{1,1}^z = 1/4 \\
 q_{2,0,*} = b_{2,0}^y = 1/8 & q_{2,1,*} = b_{2,1}^y = 1/8 & q_{2,*0} = b_{2,0}^z = 1/8 & q_{2,*1} = b_{2,1}^z = 1/8 \\
 q_{3,0,*} = b_{3,0}^y = 1/8 & q_{3,1,*} = b_{3,1}^y = 1/8 & q_{3,*0} = b_{3,0}^z = 1/8 & q_{3,*1} = b_{3,1}^z = 1/8.
 \end{array}$$

We can now give an easy expression for the dimension of the feasible region (with respect to the number of variables, $|\mathcal{J}(b)|$).

Corollary 1. *The dimension of $\mathfrak{F}(b)$ is*

$$|\mathcal{J}(b)| + |X| - |\{(x, y) \mid b_{x,y}^y > 0\}| - |\{(x, z) \mid b_{x,z}^z > 0\}|.$$

Proof. For $x \in X$, the set $Y_x := \{y \in Y \mid b_{x,y}^y > 0\}$ is non-empty, by assumption 3; the same is true for $Z_x := \{z \in Z \mid b_{x,z}^z > 0\}$. Note that $\mathcal{J}(b)$ is the disjoint union of the sets $\{x\} \times Y_x \times Z_x$, $x \in X$.

Lemmas 26 and 27 of [1] now basically complete the proof of our corollary: It implies that the dimension is equal to

$$\sum_{x \in X} (|Y_x| - 1)(|Z_x| - 1).$$

As $\mathcal{J}(b)$ is the disjoint union of the sets $\{x\} \times Y_x \times Z_x$, $x \in X$, we find this quantity to be equal to

$$|\mathcal{J}(b)| + |X| - \sum_{x \in X} (|Y_x| + |Z_x|).$$

Finally,

$$\sum_{x \in X} (|Y_x| + |Z_x|) = |\{(x, y) \mid b_{x,y}^y > 0\}| + |\{(x, z) \mid b_{x,z}^z > 0\}|,$$

which concludes the proof of the corollary. \square

Example 3 (Part 3/4). *Continuing Example 1, we find that the values for the variables $q_{0,0,0}$ and $q_{1,1,1}$ are fixed to $1/4$, whereas each of two sets of four variables $q_{2,y,z}$, $y, z \in \{0, 1\}$, and $q_{3,y,z}$, $y, z \in \{0, 1\}$, offers one degree of freedom, so the dimension should be 2. And indeed: $|\mathcal{J}| + |X| - |\{(x, y) \mid b_{x,y}^y > 0\}| - |\{(x, z) \mid b_{x,z}^z > 0\}| = 10 + 4 - 12 = 2$.*

3.2. The Objective Function and Optimality

We now discuss the objective function. The goal is to minimize

$$MI_{(x,y,z) \sim q}(x : y, z) = H_{(x,y,z) \sim q}(x) - H_{(x,y,z) \sim q}(x \mid y, z).$$

Since the distribution of x is fixed by the marginal equations, the first term in the sum is a constant, and we are left with minimizing negative conditional entropy. We start by discussing negative conditional entropy as a function on its full domain (We don't assume q to be a probability distribution, let alone to satisfy the marginal equations): $f: \mathbb{R}_+^{\mathcal{J}} \rightarrow \mathbb{R}$ (with $\mathcal{J} = \mathcal{J}(b)$ for arbitrary but fixed b)

$$f: q \mapsto \sum_{(x,y,z) \in \mathcal{J}} q_{x,y,z} \ln \left(\frac{q_{x,y,z}}{q_{*,y,z}} \right), \tag{6}$$

where we set $0 \ln(\dots) := 0$, as usual. The function f is continuous on its domain, and it is smooth on $[0, \infty]^{\mathcal{J}}$. Indeed, we have the gradient

$$\left(\nabla f(q) \right)_{x,y,z} = \partial_{x,y,z} f(q) = \ln \left(\frac{q_{x,y,z}}{q_{*,y,z}} \right), \tag{7}$$

and the Hessian

$$(Hf(q))_{(x,y,z),(x',y',z')} = \partial_{x,y,z} \partial_{x',y',z'} f(q) = \begin{cases} 0, & \text{if } (y', z') \neq (y, z) \\ \frac{-1}{q_{*,y,z}}, & \text{if } (y', z') = (y, z), x \neq x' \\ \frac{q_{*,y,z} - q_{x,y,z}}{q_{x,y,z} q_{*,y,z}}, & \text{if } (x', y', z') = (x, y, z). \end{cases} \quad (8)$$

It is worth pointing out that the Hessian, while positive semidefinite, is not positive definite, and, more pertinently, it is not in general positive definite on the tangent space of the feasible region, i.e., $r^\top Hf(q)r = 0$ is possible for $r \in \mathbb{R}^{\mathcal{J}}$ with $r(x, y, *) = r(x, *, z) = 0$ for all (x, y, z) . Indeed, if, e.g., $\mathcal{J} = X \times Y \times Z$, it is easy to see the kernel of $Hf(q)$ has dimension $|Y \times Z|$, whereas the feasible region has dimension $|X|(|Y| - 1)(|Z| - 1) = |X \times Y \times Z| - |X||Y| - |X||Z| + 1$. Hence, if $|Y \times Z| > |X||Y| + |X||Z| - 1$, then for every $q(!)$ the kernel of $Hf(q)$ the must have a non-empty intersection with the tangent space of the feasible region.

Optimality condition and boundary issues. In the case of points q which lie on the boundary of the domain, i.e., $q_{x,y,z} = 0$ for at least one triplet (x, y, z) , some partial derivatives don't exist. For y, z , denote $X_{yz} := \{x \mid (x, y, z) \in \mathcal{J}\}$. The situation is as follows.

Proposition 2. Let $q \in \mathfrak{F}$.

- (a) If there is a $(x, y, z) \in \mathcal{J}$ with $q_{x,y,z} = 0$ but $q_{*,y,z} > 0$, then f does not have a subgradient at q . Indeed, there is a feasible descent direction of f at q with directional derivative $-\infty$.
- (b) Otherwise—i.e., for all $(x, y, z) \in \mathcal{J}$, $q_{x,y,z} = 0$ only if $q_{*,y,z} = 0$ —subgradients exist. For all y, z , let $q^{y,z} \in [0, 1]^{X_{yz}}$ be a probability distribution on X_{yz} . Suppose that, for all y, z with $q_{*,y,z} > 0$,

$$q_x^{y,z} = \frac{q_{x,y,z}}{q_{*,y,z}} \quad \text{for all } x \in X_{yz}. \quad (9)$$

Then g defined by $g_{x,y,z} := \ln(q_x^{y,z})$, for all $(x, y, z) \in \mathcal{J}$, is a subgradient of f at q .

Moreover, g' is a subgradient iff there exists such a g with

- $g'_{x,y,z} \leq g$ for all $(x, y, z) \in \mathcal{J}$ with $q_{x,y,z} = 0$;
- $g'_{x,y,z} = g$ for all $(x, y, z) \in \mathcal{J}$ with $q_{x,y,z} > 0$.

Proof. For Proposition 2, let $(y, z) \in Y \times Z$ with $q_{*,y,z} > 0$, and $x \in X_{yz}$ with $q_{x,y,z} = 0$. There exist y', z' such that $q_{x,y',z'}, q_{x,y,z'} > 0$. This means that $\vec{d} := \vec{d}^{x,y,z,y',z'}$ as defined in (4) is a feasible direction. Direct calculations (written down in [12]) show that $f'(q; \vec{d}) = -\infty$. Invoking Lemma 1 yields non-existence of the subgradient.

As to Proposition 2, we prove the statement for every pair $(y, z) \in Y \times Z$ for the function

$$f_{yz} : \mathbb{R}_+^{X_{yz}} \rightarrow \mathbb{R} : q \mapsto \sum_{x \in X_{yz}} q_x \ln(q_x / q_*),$$

and then use Lemma 2.

Let us fix one pair (y, z) . If $q_{x,y,z} > 0$ for all $x \in X_{yz}$ holds, then we $f_{y,z}$ is differentiable at q , so we simply apply Remark 2.

Now assume $q_{*,y,z} = 0$. A vector $g \in \mathbb{R}^{X_{yz}}$ is a subgradient of $f_{y,z}$, iff

$$\sum_{x \in X_{yz}} r_{x,y,z} \ln(r_{x,y,z} / r_{*,y,z}) = f_{yz}(q+r) \stackrel{!}{\geq} f_{yz}(q) + g^\top r = f_{yz}(q) + \sum_{x \in X_{yz}} r_x g_x \quad (10)$$

holds for all $r \in \mathbb{R}^{\mathcal{J}}$ with $r_{x,y,z} \geq 0$ for all $(x, y, z) \in \mathcal{J}$, and $r_* > 0$.

We immediately deduce $g_x \leq 0$ for all x . Let $\varrho'_x := e^{\delta x}$ for all x , and $\varrho_x := \varrho'_x / C$, with $C := \varrho'_*$. Clearly, ϱ is a probability distribution on X_{yz} . Moreover, the difference LHS-RHS of (10) is equal to

$$D_{\text{KL}}(r/r_* \parallel \rho) + \ln C,$$

with D_{KL} denoting Kullback-Leibler divergence. From the usual properties of the Kullback-Leibler divergence, we see that this expression is greater-than-or-equal to 0 for all r , if and only if $C \geq 1$, which translates to

$$\sum_x e^{\delta x} \leq 1.$$

From this, the statements in Proposition 2 follow. \square

From the proposition, we derive the following corollary.

Corollary 2. *Suppose a minimum of f over $\mathbb{I}(b)$ is attained in a point q with $q_{x,y,z} = 0$ for a triple (x, y, z) with $b_{x,y}^y > 0$ and $b_{x,z}^z > 0$. Then $q_{u,y,z} = 0$ for all $u \in X$.*

Proof. This follows immediately from the fact, expressed in item Proposition 2 of the lemma, that a negative feasible descent direction exists at a point q which with $q_{x,y,z} = 0$ for a triple $(x, y, z) \in \mathcal{J}(b)$ with $q_{*,y,z} > 0$. \square

Based on Proposition 2 and the Karush-Kuhn-Tucker conditions, Theorem 1, we can now write down the optimality condition.

Corollary 3. *Let $q \in \mathbb{I}(b)$. The minimum of f over $\mathbb{I}(b)$ is attained in q if, and only if, (a) $q_{*,y,z} = 0$ holds whenever there is an $(x, y, z) \in \mathcal{J}(b)$ with $q_{x,y,z} = 0$; and (b) there exist*

- $\lambda_{x,y} \in \mathbb{R}$, for each (x, y) with $b_{x,y}^y > 0$;
- $\mu_{x,z} \in \mathbb{R}$, for each (x, z) with $b_{x,z}^z > 0$;

satisfying the following: For all y, z with $q_{*,y,z} > 0$,

$$\lambda_{x,y} + \mu_{x,z} = \ln\left(\frac{q_{x,y,z}}{q_{*,y,z}}\right) \quad \text{holds for all } x \in X_{yz};$$

for all y, z with $q_{*,y,z} = 0$ (but $X_{yz} \neq \emptyset$), there is a probability distribution $\varrho \in]0, 1]^{X_{yz}}$ on X_{yz} such that

$$\lambda_{x,y} + \mu_{x,z} \leq \ln(\varrho_x^{y,z}) \quad \text{holds for all } x \in X_{yz}.$$

Example 4 (Part 4/4). *Continuing Example 1, let us now find the optimal solution “by hand”. First of all, note that $q_{0,0,0} = 1/4 = q_{1,1,1}$ holds for all feasible solutions. By Corollary 2, if q is an optimal solution, since $q_{0,0,0} > 0$, we must have $q_{x,0,0} > 0$ for $x = 2, 3$; and similarly, $q_{x,1,1} > 0$ for $x = 2, 3$.*

To verify whether or not a given q is optimal, we have to find solutions $\lambda_{x,y}, \mu_{x,z}$ to the following system of equations and/or inequalities:

$$\begin{aligned}
 \lambda_{0,0} + \mu_{0,0} &= \ln(1/4 / (1/4 + q_{2,0,0} + q_{3,0,0})) \\
 \lambda_{2,0} + \mu_{2,0} &= \ln(q_{2,0,0} / (1/4 + q_{2,0,0} + q_{3,0,0})) \\
 \lambda_{3,0} + \mu_{3,0} &= \ln(q_{3,0,0} / (1/4 + q_{2,0,0} + q_{3,0,0})) \\
 \lambda_{2,0} + \mu_{2,1} &\begin{cases} = \ln(q_{2,0,1} / (q_{2,0,1} + q_{3,0,1})), & \text{if } q_{*,0,1} > 0 \\ \leq \ln(\varrho_2^{01}), & \text{if } q_{*,0,1} = 0 \end{cases} \\
 \lambda_{3,0} + \mu_{3,1} &\begin{cases} = \ln(q_{3,0,1} / (q_{2,0,1} + q_{3,0,1})), & \text{if } q_{*,0,1} > 0 \\ \leq \ln(\varrho_3^{01}), & \text{if } q_{*,0,1} = 0 \end{cases} \\
 \lambda_{2,1} + \mu_{2,0} &\begin{cases} = \ln(q_{3,1,0} / (q_{2,1,0} + q_{3,1,0})), & \text{if } q_{*,1,0} > 0 \\ \leq \ln(\varrho_2^{10}), & \text{if } q_{*,1,0} = 0 \end{cases} \\
 \lambda_{3,1} + \mu_{3,0} &\begin{cases} = \ln(q_{3,1,0} / (q_{2,1,0} + q_{3,1,0})), & \text{if } q_{*,1,0} > 0 \\ \leq \ln(\varrho_3^{10}), & \text{if } q_{*,1,0} = 0 \end{cases} \\
 \lambda_{1,1} + \mu_{1,1} &= \ln(1/4 / (1/4 + q_{2,1,1} + q_{3,1,1})) \\
 \lambda_{2,1} + \mu_{2,1} &= \ln(q_{2,1,1} / (1/4 + q_{2,1,1} + q_{3,1,1})) \\
 \lambda_{3,1} + \mu_{3,1} &= \ln(q_{3,1,1} / (1/4 + q_{2,1,1} + q_{3,1,1})).
 \end{aligned} \tag{11}$$

The ϱ 's, if needed, have to be found: both ϱ^{01} and ϱ^{10} must be probability distributions on $\{2, 3\}$.

From the conditions on the zero-nonzero pattern of an optimal solution, we can readily guess an optimal q . Let's guess wrongly, first, though: $q_{2,y,z} = q_{3,y,z} = 1/16$ for all y, z . In this case, all of the constraints in (11) become equations, and the ϱ 's don't occur. We have a system of equations in the variables $\lambda_{\cdot,\cdot}$ and $\mu_{\cdot,\cdot}$. It is easy to check that the system does not have a solution.

Our next guess for an optimal solution q is: $q_{2,0,1} = q_{3,0,1} = 0$; $q_{2,1,0} = q_{3,1,0} = 0$; $q_{2,0,0} = q_{3,0,0} = 1/8$; $q_{2,1,1} = q_{3,1,1} = 1/8$. In (11), the constraints involving $(y, z) = (0, 1), (1, 0)$ are relaxed to inequalities, with the freedom to pick arbitrary probability distributions ϱ^{01} and ϱ^{10} in the RHSs. We choose the following: $\lambda_{x,y} = \mu_{x,z} = \frac{1}{2} \ln(1/2)$ for all x, y, z . The equations are clearly satisfied. The inequalities are satisfied if we take $\varrho_0^{01} = \varrho_1^{01} = 1/2$; $\varrho_0^{10} = \varrho_1^{10} = 1/2$.

3.3. Algorithmic Approaches

We now discuss several possibilities of solving the convex program (CP): Gradient descent and interior point methods; geometric programming; cone programming over the so-called exponential cone. We'll explain the terms in the subsections.

3.3.1. Gradient Descent

Proposition 2 and Corollary 2 together with the running example make clear that the boundary is "problematic". On the one hand, the optimal point can sometimes be on the boundary. (In fact, this is already the case for the AND-gate, as computed in Appendix A.2 of [1].) On the other hand, by Corollary 2, optimal boundary points lie in a lower dimensional subset inside the boundary (codimension $\approx |X|$), and the optimal points on the boundary are "squeezed in" between boundary regions which are "infinitely strongly repellent" (which means to express that the feasible descent direction has directional derivative $-\infty$).

From the perspective of choosing an algorithm, it is pertinent that subgradients do not exist everywhere on the boundary. This rules out the use of algorithms which rely on evaluating (sub-)gradients on the boundary, such as projected (sub-)gradient descent. (And also generic active set

and sequential quadratic programming methods (We refer the interested reader to [13] for background on these optimization methods); the computational result in Section 4 illustrate that.

Due to the huge popularity of gradient descent, the authors felt under social pressure to present at least one version of it. Thus, we designed an ad-hoc quick-and-dirty gradient descent algorithm which does its best to avoid the pitfalls of the feasible region: it’s boundary. We now describe this algorithm.

Denote by A the matrix representing the LHS of the equations in (CP); also reduce A by removing rows which are linear combinations of other rows. Now, multiplication by the matrix $P := A^T(A^T A)^{-1}A$ amounts to projection onto the tangent space $\{d \mid Ad = 0\}$ of the feasible region, and $P\nabla f(q)$ is the gradient of f in the tangent space, taken at the point q .

The strategy by which we try to avoid the dangers of approaching the boundary of the feasible region in the “wrong” way is by never reducing the smallest entry of the current iterate q by more than 10%. Here is the algorithm.

Algorithm 1: Gradient Descent

- 1 Construct the matrix A
 - 2 Compute $P := A^T(A^T A)^{-1}A$
 - 3 Initialize q to a point in the interior of the feasible region
 - 4 **repeat**
 - 5 Compute $f(q)$ **if** $f(q)$ better than all previous solutions **then**
 - 6 store q
 - 7 Compute the gradient $\nabla f(q)$
 - 8 Compute the projection of the gradient $g := P\nabla f(q)$
 - 9 Determine a step size η , ensuring $q_{xyz} > \eta g_{xyz}$ for all xyz
 - 10 Update $q = q - \eta g$
 - 11 **until** stopping criterion is reached
-

There are lots of challenges with this approach, not the least of which is deciding on the step size η . Generally, a good step size for gradient descent is 1 over the largest eigenvalue of the Hessian—but the eigenvalues of the Hessian tend to infinity.

The stopping criterion is also not obvious: we use a combination of the norm of the projected gradient, the distance to the boundary, and a maximum of 1000 iterations.

None of these decisions are motivated by careful thought.

3.3.2. Interior Point Methods

Using Interior Point Methods (IPMs) appears to be the natural approach: While the iterates can converge to a point on the boundary, none of the iterates actually lie on the boundary, and that is an inherent property of the method (not a condition which you try to enforce artificially as in the gradient descent approach of the previous section). Consequently, problems with gradients, or even non-existing subgradients, never occur.

Even here, however, there are caveats involving the boundary. Let us consider, as an example, the simple barrier approach sketched in Section 2.1 (page 4). The analysis of this method (see [11]) requires that the function $F: q \mapsto tf(q) - \sum_{xyz} \ln(q_{xyz})$ be *self-concordant*, which means that, for some constant C , for all q, h

$$D^3F(q)[h, h, h] \leq C \cdot (h^T HF(q)h)^{3/2}, \tag{12}$$

where D^3 denotes the tensor of third derivatives. The following is proven in [12]:

Proposition 3. Let $n \geq 2$, and consider the function

$$F: q \mapsto t \sum_{x=1}^n q_x \ln(q_x/q_*) - \sum_x \ln(q_x)$$

There is no C and no t such that (12) holds for all $q \in [0, \infty]^n$ and all h .

The proposition explains why, even for some IPMs, approaching the boundary can be problematic. We refer to [12] for more discussions about self-concordancy issues of the (CP).

Corollary 4. The complexity of the interior point method via self-concordant barrier for solving (CP) is $O(M \cdot \log^{1/\epsilon})$, where M is the complexity of computing the Newton step (which can be done by computing and inverting the Hessian of the barrier)

Proof. This follows immediately from the fact that a 3-self concordant barrier exists for (CP) (see [12]) and the complexity analysis of barrier method in [11], Chapter 11. \square

Still, we find the IPM approach (with the commercial Interior Point software “Mosek”) to be the most usable of all the approaches which we have tried.

3.3.3. Geometric Programming

Geometric Programs form a sub-class of Convex Programs; they are considered to be more easily solvable than general Convex Programs: Specialized algorithms for solving Geometric Programs have been around for a half-century (or more). We refer to [14] for the definition and background on Geometric Programming. The Langrange dual of (CP) can be written as the following Geometric Program in the variables $\lambda \in \mathbb{R}^{X \times Y}$, $\mu \in \mathbb{R}^{X \times Z}$ (for simplicity, assume that all the right-hand-sides b^* are strictly positive):

$$\begin{aligned} & \text{minimize} && \sum_{(x,y) \in X \times Y} b_{x,y}^y \lambda_{x,y} + \sum_{(x,z) \in X \times Z} b_{x,z}^z \mu_{x,z} \\ & \text{subject to} && \ln \left(\sum_{x \in X} \exp(\lambda_{xy} + \mu_{xz}) \right) \leq 0 \quad \text{for all } (y,z) \in Y \times Z. \end{aligned} \tag{GP}$$

3.3.4. Exponential Cone Programming

Cone Programming is a far-reaching generalization of Linear Programming, which may contain so-called *generalized inequalities*: For a fixed closed convex cone \mathcal{K} , the generalized inequality “ $a \leq_{\mathcal{K}} b$ ” simply translates into $b - a \in \mathcal{K}$. There is a duality theory similar to that for Linear Programming.

Efficient algorithms for Cone Programming exist for some closed convex cones; for example for the *exponential cone* [15], \mathcal{K}_{exp} , which is the closure of all triples $(r, p, q) \in \mathbb{R}^3$ satisfying

$$q > 0 \text{ and } qe^{r/q} \leq p.$$

For $q > 0$, the condition on the right-hand side is equivalent to $r \leq q \ln(p/q)$, from which it can be easily verified that the following “Exponential Cone Program” computes (CP). The variables are $r, q, p \in \mathbb{R}^{X \times Y \times Z}$.

$$\begin{aligned}
 & \text{maximize} && \sum_{x,y,z} r_{x,y,z} \\
 & \text{subject to} && q_{x,y,*} = b_{x,y}^y && \text{for all } (x,y) \in X \times Y \\
 & && q_{x,*,z} = b_{x,z}^z && \text{for all } (x,z) \in X \times Z \\
 & && q_{*,y,z} - p_{x,y,z} = 0 && \text{for all } (x,y,z) \in X \times Y \times Z \\
 & && (r_{x,y,z}, q_{x,y,z}, p_{x,y,z}) \in \mathcal{K}_{\text{exp}} && \text{for all } (x,y,z) \in X \times Y \times Z.
 \end{aligned} \tag{EXP}$$

The first two constraints are just the marginal equations; the third type of equations connects the p -variables with the q -variables, and the generalized inequality connects these to the variables forming the objective function.

There are in fact several ways of modeling (CP) as an Exponential Cone Program; here we present the one which is most pleasant both theoretically (the duality theory is applicable) and in practice (it produces the best computational results). For the details, as well as for another model, we refer to [12].

4. Computational Results

In this section, we present the computational results obtained by solving Bertschinger et al.’s Convex Program (CP) on a large number of instances, using readily available software out-of-the-box. First we discuss the problem instances, then the convex optimization solvers, then discuss the results. Detailed tables are in the Appendix A.

4.1. Data

In all our instances, the vectors b^y and b^z are marginals computed from an input probability distribution p on $X \times Y \times Z$. Occasionally, “noise” (explained below) in the creation p leads to the phenomenon that the sum of all probabilities is not 1. This was not corrected before computing b^y and b^z , as it is irrelevant for the Convex Program (CP).

We have three types of instances, based on (1) “gates”—the “paradigmatic examples” listed in Table 1 of [1]; (2) Example 31 and Figure A.1 of [1]; (3) discretized 3-dimensional Gaussians.

- (1) Gates. The instances of the type (1) are based on the “gates” (RDN, UNQ, XOR, AND, RDNXOR, RDNUNQXOR, XORAND) described Table 1 of [1]:

RDN $X = Y = Z$ uniformly distributed.

UNQ $X = (Y, Z)$, Y, Z independent, uniformly distributed in $\{0, 1\}$.

XOR $X = Y \text{ XOR } Z$, Y, Z independent, uniformly distributed in $\{0, 1\}$.

AND $X = Y \text{ AND } Z$, Y, Z independent, uniformly distributed in $\{0, 1\}$.

RDNXOR $X = (Y_1 \text{ XOR } Z_1, W)$, $Y = (Y_1, W)$, $Z = (Z_1, W)$, Y_1, Z_1, W independent, uniformly distributed in $\{0, 1\}$.

RDNUNQXOR $X = (Y_1 \text{ XOR } Z_1, (Y_2, Z_2), W)$, $Y = (Y_1, Y_2, W)$; $Z = (Z_1, Z_2, W)$, Y_1, Y_2, Z_1, Z_2, W independent, uniformly distributed in $\{0, 1\}$.

XORAND $X = (Y \text{ XOR } Z, Y \text{ AND } Z)$, Y, Z independent, uniformly distributed in $\{0, 1\}$.

- Each gate gives rise to two sets of instances: (1a) the “unadulterated” probability distribution computed from the definition of the gate (see Table A1); (1b) empirical distributions generated by randomly sampling from \mathbf{W} and computing (x, y, z) . In creating the latter instances, we incorporate “noise” by perturbing the output randomly with a certain probability. We used 5 levels of noise, corresponding to increased probabilities of perturbing the output. For example, AND 0 refers to the empirical distribution of the AND gate without perturbation (probability = 0), AND 4 refers to the empirical distribution of the AND gate with output perturbed with probability 0.1. The perturbation probabilities are: 0, 0.001, 0.01, 0.05, 0.1. (See Tables A2 and A3).
- (2) Example-31 instances. In Appendix A.2 of their paper, Bertschinger et al. discuss the following input probability distribution: $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are independent uniformly random in $\{0, 1\}$. They present

this as an example that the optimization problem can be “ill-conditioned”. We have derived a large number of instances based on that idea, by taking $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ uniformly distributed on $X \times Y \times Z$, with $|X|$ ranging in $\{2, \dots, 5\}$ and $|Y| = |Z|$ in $\{|X|, \dots, 5 \cdot |X|\}$. These instances are referred to as “independent” in Table A4. We also created “noisy” versions of the probability distributions by perturbing the probabilities. In the results, we split the instances into two groups according to whether the fraction $|Y|/|X|$ is at most 2 or greater than 2. (The rationale behind the choice of the sizes of $|X|$, $|Y|$, $|Z|$ is the fact mentioned in Section 3.2, that the Hessian has a kernel in the tangent space if the ratio is high.)

- (3) Discretized Gaussians. We wanted to have a type of instances which was radically different from the somewhat “combinatorial” instances (1) and (2). We generated (randomly) twenty 3×3 covariance matrices between standard Gaussian random variables \mathbf{X} , \mathbf{Y} , and \mathbf{Z} . We then discretized the resulting continuous 3-dimensional Gaussian probability distribution by integrating numerically (We used the software Cuba [16] in version 4.2 for that) over boxes $[0, 1]^3$, $[0, 0.75]^3$, $[0, 0.5]^3$, $[0, 0.25]^3$; and all of their translates which held probability mass at least 10^{-20} .

We grouped these instances according to the number of variables, see Table A5: The instances in the “Gauss-1” group are the ones with at most 1000 variables; “Gauss-2” have between 1000 and 5000; “Gauss-3” have between 5000 and 250,000, “Gauss-4” between 25,000 and 75,000.

4.2. Convex Optimization Software We Used

For our computational results, we made an effort to use as many software toolboxes as we could get our hands on. (The differences in the results are indeed striking.) Most of our software was coded in the Julia programming language (version 0.6.0) using the “MathProgBase” package see [17] which is part of JuliaOpt. (The only the CVXOPT interior point algorithm and for the Geometric Program did we use Python.) On top of that, the following software was used.

- CVXOPT [18] is written and maintained by Andersen, Dahl, and Vandenberghe. It transforms the general Convex Problems with nonlinear objective function into an epigraph form (see Section 2.1), before it deploys an Interior Point method. We used version 1.1.9.
- Artelys Knitro [19] is an optimization suite which offers four algorithms for general Convex Programming, and we tested all of them on (CP). The software offers several different convex programming solvers: We refer by “Knitro_Ip” to their standard Interior Point Method [20]; by “Knitro_IpCG” to their IPM with conjugate gradient (uses projected cg to solve the linear system [21]); “Knitro_As” is their Active Set Method [22]; “Knitro_SQP” designates their Sequential Quadratic Programming Method [19]. We used version 10.2.
- Mosek [23] is an optimization suite which offers algorithms for a vast range of convex optimization problems. Their Interior Point Method is described in [24,25]. We used version 8.0.
- Ipopt [20] is a software for nonlinear optimization which can also deal with non-convex objectives. At its heart is an Interior Point Method (as the name indicates), which is enhanced by approaches to ensure convergence even in the non-convex case. We used version 3.0.
- ECOS. We are aware of only two Conic Optimization software toolboxes which allow to solve Exponential Cone Programs: ECOS and SCS. ECOS is a lightweight numerical software for solving convex cone programs [26], using an Interior Point approach. We used the version from Nov 8, 2016.
- SCS [27,28] stands for Splitting Conic Solver. It is a numerical optimization package for solving large-scale convex cone problems. It is a first-order method, and generally very fast. We used version 1.2.7.

4.3. Results

We now discuss the computational results of every solver. The tables in Appendix A give three data points for each pair of instance (group) and solver: Whether the instance was solved, or,

respectively, which fraction of the instances in the group were solved (“solved”); the time (“time”) or, respectively, average time “avg. tm” needed (All solvers had a time limit 10,000.0 s. We used a computer server with Intel(R) Core(TM) i7-4790K CPU (4 cores) and 16GB of RAM) to solve it/them. The third we call the “optimality measure”: When a solver reports having found (optimal) primal and dual solutions, we computed:

- the maximum amount by which any of the marginal equations is violated;
- the maximum amount by which any of the nonnegativity inequities is violated;
- the maximum amount by which the inequality “ $A^T \lambda \leq g$ ” in Theorem 1 is violated;
- the maximum amount by which the complementarity $|x_j| |A^T \lambda)_j - g_j|$ is violated.

Of all these maxima we took that maximum to have an indication of how reliable the returned solution is, with respect to feasibility and optimality. In the tables, the maximum is then taken over all the instances in the group represented in each row of the table. We chose this “worst-case” approach to emphasize the need for a *reliable* software.

4.3.1. Gradient Descent

The Gradient Descent algorithm sketched in the previous section finds its limits when the matrix P becomes too large to fit into memory, which happens for the larger ones of the Gaussian instances.

Even before that, the weakness of the algorithm are clear in the fact that the best solution it finds is often not optimal (the other methods produce better solutions). The running times are mid-field compared to the other methods.

It is probable that a more carefully crafted gradient descent would perform better, but the conceptual weaknesses of the approach remain.

4.3.2. Interior Point algorithms

Cvxopt It terminated with optimal solution on all the noiseless discrete gates except “RDN” see Tables A1–A3. But whenever the noise was added it started failing except on “XOR”. So as the number of samples increased even when decreasing the noise it still failed. The optimality measure was always bad except on some of the noiseless gates. Mainly the dual solution was weakly feasible on the solved ones. It bailed out 56% due to KKT system problems and the rest was iteration and time limit.

For the uniformly binary gates again Cvxopt failed on the noisy distributions with 34% success on Noisy 1 and 0% on Noisy 2. It solved all the independent correctly with acceptable optimality measure.

For the Gaussian, it totally failed with 40% computational problems in solving the KKT system and 38% time limit and the rest were iteration limit. The solver was slowest compared to others and required high memory to store the model.

Knitro_Ip It terminated with optimal solution on all the discrete gates with good optimality measure see Tables A1–A4. But for the Gaussian, it failed most of the time. It solved only 5% of all Gaussian instances where 75% of the solved have less than 1000 variable Table A5. 25% of the unsolved instances couldn’t get a dual feasible solution. The rest stopped due to iteration or time limit but still they had either infeasible dual or the KKT conditions were violated.

Kintro_Ip was a little bit slower than Mosek on most of the discrete gates. On the Gaussian, it very slow. For most of the instances, Mosek had its optimality measure 1000 times better than Knitro_Ip. Even though it worked well for the discrete gates we can’t rely on it for this problem since it is not stable with the variables number.

Knitro IpCG. It terminated with optimal solution on all the discrete gates except “XOR” and “RDNXOR” see Tables A2 and A3. For those two particular gates, it reached the iteration limit. For the Gaussian, it did a little better than Knitro_Ip since the projected conjugate gradient helps with large scale problems. It was able to solve only 6.2% none of which had less than 1000 variables and with 50% having more than 8000 variables see Table A5. 25% of the unsolved instances couldn’t get a dual feasible solution. The rest stopped due to iteration or time limit with similar optimality measure violation as Knitro_Ip.

Knitro_IpCG was considerably slower than Mosek and Knitro_Ip. And it had worse optimality measure than Mosek on most of its solved instances. Since Knitro_IpCG couldn't solve a big set of instances this solver is unreliable for this problem.

Mosek terminated with reporting an optimal solution almost on all instances. For the gates, with sufficient optimality measure see Tables A1–A4. For the Gaussian gates, it terminated optimally for 70% of the instances see Table A5. 25% bailed out mainly since the gap between the primal and dual is negligible with the dual solution and primal are weakly feasible. Most of the latter instances had more than 10,000 variables. For the last 5% Mosek reached its iteration limit.

For all discrete gates Mosek terminated within milliseconds except "RDNUNQXOR" it terminated in at most 1 s. For the Gaussians only those with more than 25,000 variables took up to 5 s to be solved, the rest was done in less than a second.

Ipopt failed on the unadulterated "AND", "UNQ", and "XORAND" distributions since the optimal solution lays on the boundary and Ipopt hits the boundary quit often. Also, it couldn't solve any of the noiseless with sampling "AND" and "UNQ" gates, but managed to solve 30% of those of "XORAND". When the noise was applied, it worked 90% with some noisy "RDNXOR" gates, but bailed out on "RDNUNQXOR" gate with 10%, 5%, and 1%. Note that the optimality measure was highly violated even when Ipopt terminates optimally see Tables A1–A4 The violation mainly was in the dual feasibility and for the instances which had the solution on the boundary.

With the Gaussians, Ipopt couldn't solve any instance see Table A5. In 55% of the instances it terminated with computational errors or search direction being to small. 25% terminated with iteration limit and the rest with time limit. Same as for discrete case the optimality measure was not good. For discrete gates Ipopt was as fast a Mosek. Overall Ipopt is an unreliable solver for this problem.

4.3.3. Geometric Programming: CVXOPT

CVXOPT has a dedicated Geometric Programming interface, which we used to solve the Geometric Program (GP). The approach cannot solve a single instance to optimality: CVXOPT always exceeds the limit on the number of iterations. Hence, we have not listed it in the tables.

4.3.4. Exponential Cone Programming: ECOS & SCS

SCS failed on each and every instance. For ECOS, however, on average the running times were fast, but the results were mildly good. For types 1 and 2 instances, ECOS was most of the time the fastest solver. It terminated optimally for all the instances of types 1 and 2.

ECOS terminated suboptimal on 90% of the Gaussian instances. On the rest of the Gaussian gates it ran into numerical problems, step size became negligible, and terminated with unfeasible solutions.

ECOS handled types 1 and 2 instances pretty well. It was mostly slow on the Gaussian instances. For example, some Gaussian instances took ECOS more than seven hours to terminate sub-optimally. Modeling the problem as an exponential cone programming seems to be a good choice. ECOS vigorous performance makes it a reliable candidate for this optimization problem.

4.3.5. Miscellaneous Approaches

We include the following in the computational results for completeness. From what we discussed in the previous section, it is no surprise at all that these methods perform badly.

Knitro_SQP failed on the unadulterated "AND" and "RDNUNQAND" distributions see Table A1. When the noise was added to the discrete gates it couldn't solve more than 35% per gate see Tables A2 and A3. It didn't have better optimality measure than Mosek except on "RDN 1" which is can't be built on since Knitro_SQP solved only 8% of those instances. Note that on the unsolved instances, Knitro_SQP was the only solver which gave wrong answers (30% of the unsolved) i.e., claimed that the problem is infeasible.

Similarly, on Gaussian it failed in 78% of the instances due to computational problems and the rest were iteration or time limit see Table A5. This confirms that such type of algorithms is unsuitable for our problem; see Section 3.

Knitro_As. It failed on all the instances, trying to evaluate the Hessian at boundary points.

5. Conclusions

A closer look at the Convex Program proposed by Bertschinger et al. [1] reveals some subtleties which makes the computation of the information decomposition challenging. We have tried to solve the convex program using a number of software packages out-of-the-box (including, BTW, [29]).

Two of the solvers, namely, Mosek and ECOS work very satisfactorily. Even though, ECOS sometimes was 1000 times faster than Mosek, on some of types 1 and 2 instances, the time difference was no more than 5 s making this advantage rather dispensable. Nevertheless, on these instances, Mosek had better optimality measures. Note that on the hard problems of type 1, namely, “RDNUNQXOR”, Mosek was faster than ECOS see Figure 1.

On the other hand, ECOS was slower than Mosek on the Gaussian instances especially when the (CP) had a huge number of variables. For these instances, the time difference is significant (hours) see Figure 2, which is rather problematic for ECOS.

Hence each of the two solvers has its pros and cones. This means that using both solvers is an optimal strategy when approaching the problem. One suggestion would be to use ECOS as a surrogate when Mosek fails to give a solution. Earlier ad-hoc approaches, which were based on CVXOPT and on attempts to “repair” a situation when the solver failed to converge, appear to be redundant.

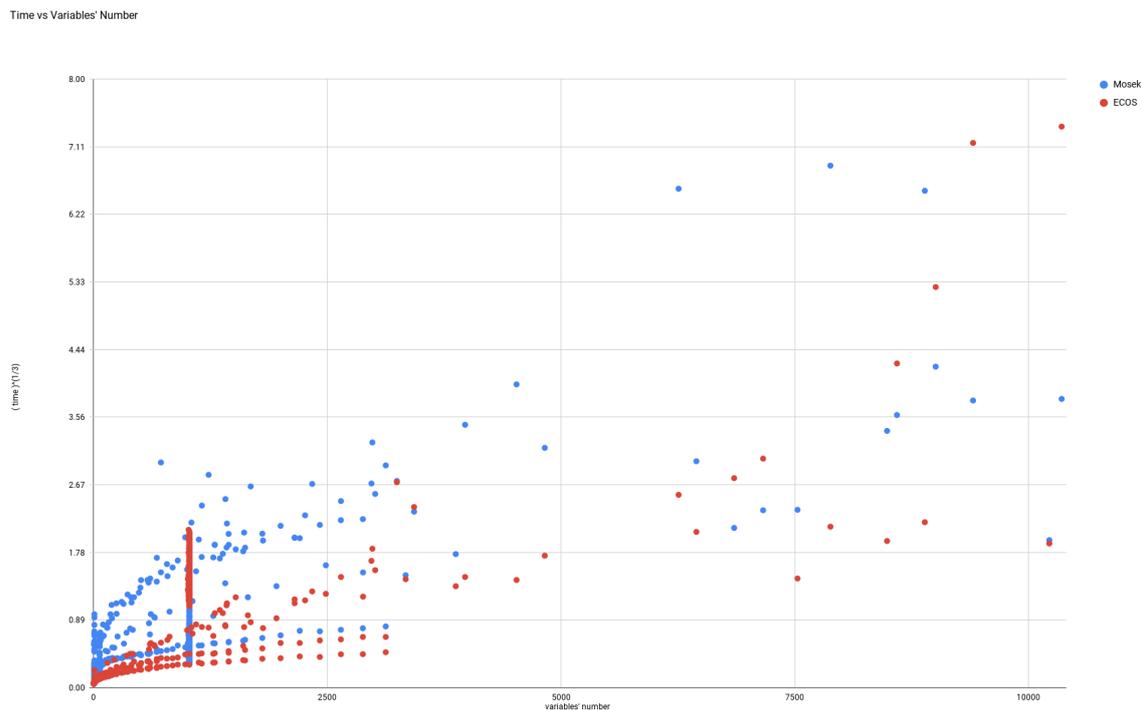


Figure 1. Comparison Between the running times of Mosek and ECOS for problems whose number of variables is at below 10,500.

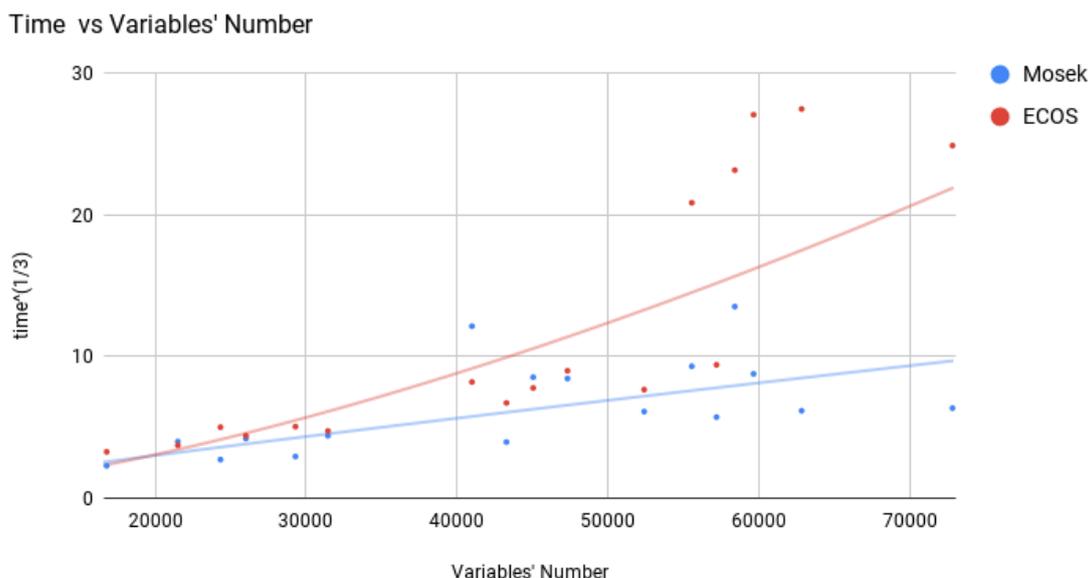


Figure 2. Comparison Between the running times of Mosek and ECOS for problems whose number of variables is at least 16,000.

Supplementary Materials: The source code in Julia used for the computations can be found here: github.com/dot-at/BROJA-Bivariate-Partial_Information_Decomposition.

Acknowledgments: This research was supported by the Estonian Research Council, ETAG (*Eesti Teadusagentuur*), through PUT Exploratory Grant #620. We also gratefully acknowledge funding by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

Author Contributions: RV+DOT developed the first algorithms and wrote the code. DOT+AM contributed the propositions and proofs, AM performed the computations and created the tables.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Tables of Instances and Computational Results

Table A1. Results for type (1a) instances: Gates—unadulterated probability distributions.

Instance	CVXOPT			Knitro_Ip			Knitro_IpCG			Knitro_As			Knitro_SQP			Mosek			Ipopt			ECOS			GD		
	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	
XOR	y	1	0.06	y	0.7	0.31	y	98	0.8	n				2	0.63	y	0.18	0.28	y	0.15	1.5	y	0.03	0.001	25		
AND	y	0.5	21e ⁷	y	30	0.5	y	52	13	n						y	0.06	0.15	y	0.09	10	y	0.03	0.01	25		
UNQ	y	0.5	0.02	y	0.1	81e ⁻⁴	y	0.17	0.04	n			y	0.8	21e ⁻¹²	n	0.07	1e ⁻⁴	n			y	0.02	0.01	14		
RDN	y	0.4	0.007	y	0.07	1e ⁻⁷	y	0.09	0.38	n			y	0.1	41e ⁻¹²	y	0.03	0.025	n			y	0.02	0.002	15		
XORAND	y	0.6	21e ⁶	y	0.2	0.53	y	0.9	0.14	n			y	0.6	1e ⁻⁷	y	0.09	0.2	n			y	0.03	0.005	14		
RDNXOR	y	3	0.12	y	0.2	0.06	y	271	0.3	n			y			y	0.2	2e ⁻⁴	y	0.2	2e ⁴	y	0.05	0.002	24		
RDNUNQXOR	y	14	0.4	y	1.3	0.001	y	537	0.09	n			n			y	0.01	6e ⁻⁷	y	0.5	150	y	0.16	0.008	77		

Table A2. Results for type (1b) instances: Gates (XOR, AND, UNQ, RDN) with noise.

Instance	CVXOPT			Knitro_Ip			Knitro_IpCG			Knitro_As			Knitro_SQP			Mosek			Ipopt			ECOS			GD
	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	avg. tm 10^{-2}
XOR 0	100	0.01	3e6	100	0.9	0.46	100	150	1.06	0	30	4	0.87	100	4	0.004	100	0.1	2e6	100	0.04	0.01	28		
XOR 1	100	2.2	3e6	100	1	0.98	89	254	2.02	0	17	4	0.67	100	1.9	0.0066	100	0.2	3e6	100	0.03	0.01	35		
XOR 2	100	2	3e6	100	1	0.92	63	259	1	0	26	9	0.91	100	12	0.0053	100	0.3	2e6	100	0.04	0.01	36		
XOR 3	99	2	4e6	100	1	0.96	70	363	1	0	27	7	0.99	100	7	0.0056	100	0.2	2e6	100	0.03	0.01	36		
XOR 4	100	1	4e6	100	1.3	1	98	40	1	0	20	13	0.98	100	8	0.0052	100	0.1	1e6	100	0.03	0.01	32		
AND 0	100	300	0.6	100	40	0.7	100	76	22.5	0	0	0	0	0	0.07	0.41	0	100	0.07	0.41	0	100	0.03	0.01	27
AND 1	65	2	2e7	100	1	2.9	100	17	3.2	0	0	0	0	0	0.9	24	100	0.3	2e6	100	0.04	0.01	42		
AND 2	67	3	1e6	100	2	1.7	100	4.9	4	0	0	0	0	0	1.2	0.11	100	0.3	1e7	100	0.04	0.01	40		
AND 3	51	3	2e7	100	1.9	1.9	100	2	18	0	0	0	0	0	1.4	0.04	100	4	2e6	100	0.04	0.01	39		
AND 4	54	2	2e7	100	1	5.4	100	2	9	0	0	0	0	0	0.5	41	100	0.5	4e4	100	0.04	0.01	40		
UNQ 0	100	0.4	2e4	100	0.4	3e-3	100	0.5	0.09	0	100	1.4	6e-12	100	0.04	1e-4	0	100	0.04	1e-4	100	0.02	0.007	16	
UNQ 1	100	4	4e6	100	1	6	100	1.9	9	0	33	6	5	100	0.6	0.036	100	0.7	3e5	100	0.07	0.01	42		
UNQ 2	100	5	5e6	100	2	0.04	100	2	0.84	0	26	13	0.46	100	0.69	0.12	100	0.7	4e5	100	0.07	0.008	44		
UNQ 3	100	5	7e7	100	2	6	100	1	40	0	38	9	10	100	0.7	0.022	100	0.6	4e5	100	0.07	0.004	44		
UNQ 4	20	4	0.09	100	1.5	0.04	100	1	21	0	18	8	2.6	100	0.6	0.0019	100	0.7	3e5	100	0.08	0.005	42		
RDN 0	-	2	2e6	100	0.4	6e-4	100	0.4	0.51	0	100	0.6	1e-12	100	1	0.0029	30	0.5	1e5	100	0.02	0.01	16		
RDN 1	0	1	2.8	100	1.2	2.8	0	0	0	0	8	2	2e-12	100	0.3	0.005	100	0.3	2e7	100	0.04	0.01	41		
RDN 2	0	100	1.9	6.7	100	1.9	6.2	0	0	0	14	4.5	2e-10	100	1.1	0.01	100	0.5	2e6	100	0.04	0.01	40		
RDN 3	0	100	1.7	4.1	100	1.8	9.8	0	0	0	35	4	2.7	100	0.4	2e-4	100	0.6	4e6	100	0.05	0.01	41		
RDN 4	1	12	2e6	100	0.9	0.07	100	0.8	1	0	3	3	2e-6	100	0.2	1e-5	100	0.4	4e6	100	0.04	0.008	41		

Table A3. Results for type (1b) instances: Gates (XORAND, RDNXOR, RDNUNQXOR) with noise.

Instance	CVXOPT			Knitro_Ip			Knitro_IpCG			Knitro_As			Knitro_SQP			Mosek			Ipopt			ECOS			GD
	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	avg. tm 10^{-2}	Opt.meas. 10^{-6}	% solved	time (10^{-2})	opt.meas. (10^{-6})	time (10^{-2})
XORAND 0	100	0.6	20	100	0.6	0.74	100	0.9	0.2	0	100	2	2e-6	100	0.1	0.4	0	100	0.03	0.01	27				
XORAND 1	94	6	1e7	100	1.1	5	100	5	6.2	0	35	9	4.9	100	1	0.5	100	0.6	8e5	100	0.08	18	43		
XORAND 2	67	8	12.6	100	2	6.2	100	10	8.2	0	15	22	5.4	100	1.3	0.4	100	0.8	1e5	100	0.09	0.01	42		
XORAND 3	12	100	2	5.7	100	2	8.7	0	0	0	15	16	6.9	100	1.3	2.2	100	1	6e4	100	0.1	0.03	42		
XORAND 4	0	100	2	7.2	100	3	8	0	0	0	93	2	100	0.7	40	100	0.8	4e4	100	0.1	0.04	43			
RDNXOR 0	100	3	3.11	100	1	0.46	70	407	1	0	30	6	0.23	100	1	2e-9	100	0.2	2e4	100	0.06	0.01	27		
RDNXOR 1	0	100	3.5	18.2	100	2	30	0	0	0	0	0	0	100	25	5	1	261	0.016	100	0.3	0.01	40		
RDNXOR 2	1	190	170	100	9	60	100	3	250	0	0	0	0	100	250	50	90	10	1.9	100	0.4	0.01	41		
RDNXOR 3	0	100	8	93	100	3	30	0	0	0	1	1	0	100	15	0.01	99	4.7	0.02	100	0.3	0.01	40		
RDNXOR 4	1	145	2e7	100	6	90	100	5	31	0	0	0	0	100	15	0.01	100	5	0.02	100	0.4	0.01	43		
RDNUNQXOR 0	90	25	3e6	100	2	0.34	100	636	1	0	0	0	0	100	6	0.2	100	0.7	234	100	0.3	0.008	75		
RDNUNQXOR 1	0	100	57	297	100	34	107	0	0	0	0	0	0	100	269	0.4	0	0	0	100	200	0.01	1156		
RDNUNQXOR 2	0	100	110	364	100	60	128	0	0	0	0	0	0	100	483	4	0	0	0	100	195	0.01	1170		
RDNUNQXOR 3	0	100	130	561	100	91	597	0	0	0	0	0	0	100	307	20	0	0	0	100	224	0.01	1140		
RDNUNQXOR 4	0	100	129	422	100	132	1e4	0	0	0	0	0	0	100	388	0.003	92	437	207	100	555	0.024	1160		

Table A4. Results for type (2) instances: Example-31.

Instance	CVXOPT			Mosek			Ipopt			ECOS			GD
	% solved	time (10^{-2})	opt. meas. (10^{-6})	% solved	time (10^{-2})	opt. meas. (10^{-6})	% solved	time (10^{-2})	opt. meas. (10^{-6})	% solved	time (10^{-2})	opt. meas. (10^{-6})	time (10^{-2})
Independent 1	100	28	1.15	100	4	91e-4	100	18	31e-3	100	0.5	0.01	100
Independent 2	100	255	0.2	100	22	0.003	92	437	207	100	2	0.02	802
Noisy 1	34	113	21e7	100	87	0.008	38	6.2	21e3	100	2.8	0.014	100
Noisy 2	0	100	648	100	648	0.011	7	602	21e-3	100	8	0.016	813

Table A5. Results for type (3) instances: Discretized 3-dimensional Gaussians.

Instance	CVXOPT			Knitro_Ip			Knitro_IpCG			Knitro_As			Knitro_SQP			Mosek			Ipopt			ECOS			GD	
	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	% solved	time (10 ⁻²)	opt. meas. (10 ⁻⁶)	time (10 ⁻²)	
Gauss-1	0			21	820	4	0								71	0.27	0.4	0	100	4.2	0.19	1940				
Gauss-2	0			2.7	950	10	5	1020	300	0					75	7.6	4	0	96	52	1.06	1890				
Gauss-3	0			0			11	2170	900	0					65	551	2	0	96	632	6	-				
Gauss-4	0			0			7	2560	2000	0					57	4e ⁴	60	0	75	3e ⁵	400	-				

References

- Bertschinger, N.; Rauh, J.; Olbrich, E.; Jost, J.; Ay, N. Quantifying unique information. *Entropy* **2014**, *16*, 2161–2183.
- Williams, P.L.; Beer, R.D. Nonnegative decomposition of multivariate information. *arXiv* **2010**, arXiv:1004.2515.
- Griffith, V.; Koch, C. Quantifying synergistic mutual information. In *Guided Self-Organization: Inception*; Springer: New York, NY, USA, 2014; pp. 159–190.
- Harder, M.; Salge, C.; Polani, D. Bivariate measure of redundant information. *Phys. Rev. E* **2013**, *87*, 012130.
- Bertschinger, N.; Rauh, J.; Olbrich, E.; Jost, J. Shared information—New insights and problems in decomposing information in complex systems. In *Proceedings of the European Conference on Complex Systems, Brussels, Belgium, 2–7 September 2012*; Springer: Cham, Switzerland, 2013; pp. 251–269.
- Nemirovski, A. Efficient Methods in Convex Programming. Available online: http://www2.isye.gatech.edu/~nemirovs/Lect_EMCO.pdf (accessed on 6 October 2017).
- Wibral, M.; Priesemann, V.; Kay, J.W.; Lizier, J.T.; Phillips, W.A. Partial information decomposition as a unified approach to the specification of neural goal functions. *Brain Cognit.* **2017**, *112*, 25–38.
- Ruszczynski, A.P. *Nonlinear Optimization*; Princeton University Press: Princeton, NJ, USA, 2006; Volume 13.
- Bubeck, S.; Eldan, R. The entropic barrier: A simple and optimal universal self-concordant barrier. *arXiv* **2014**, arXiv:1412.1587.
- Bubeck, S.; Eldan, R. The entropic barrier: A simple and optimal universal self-concordant barrier. In *Proceedings of the 28th Conference on Learning Theory, Paris, France, 3–6 July 2015*.
- Boyd, S.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
- Makkeh, A. Applications of Optimization in Some Complex Systems. Ph.D. Thesis, University of Tartu, Tartu, Estonia, 2017.
- Nocedal, J.; Wright, S. *Numerical Optimization*; Springer: New York, NY, USA, 2006.
- Boyd, S.; Kim, S.J.; Vandenberghe, L.; Hassibi, A. A tutorial on geometric programming. *Optim. Eng.* **2007**, *8*, 67–127.
- Chares, R. Cones and interior-point algorithms for structured convex optimization involving powers and exponentials, Doctoral dissertation, UCL-Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2009.
- Hahn, T. CUBA—A library for multidimensional numerical integration. *Comput. Phys. Commun.* **2005**, *168*, 78–95. [[hep-ph/0404043](https://arxiv.org/abs/hep-ph/0404043)].
- Lubin, M.; Dunning, I. Computing in operations research using Julia. *INFORMS J. Comput.* **2015**, *27*, 238–248.
- Sra, S.; Nowozin, S.; Wright, S.J. *Optimization for Machine Learning*; MIT Press: Cambridge, MA, USA, 2012.
- Byrd, R.; Nocedal, J.; Waltz, R. KNITRO: An Integrated Package for Nonlinear Optimization. In *Large-Scale Nonlinear Optimization*; Springer: New York, NY, USA, 2006; pp. 35–59.
- Waltz, R.A.; Morales, J.L.; Nocedal, J.; Orban, D. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Math. Program.* **2006**, *107*, 391–408.
- Byrd, R.H.; Hribar, M.E.; Nocedal, J. An interior point algorithm for large-scale nonlinear programming. *SIAM J. Optim.* **1999**, *9*, 877–900.
- Byrd, R.H.; Gould, N.I.; Nocedal, J.; Waltz, R.A. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Math. Program.* **2004**, *100*, 27–48.

23. ApS, M. *Introducing the MOSEK Optimization Suite 8.0.0.94*. Available online: http://docs.mosek.com/8.0/pythonfusion/intro_info.html (accessed on 6 October 2017).
24. Andersen, E.D.; Ye, Y. A computational study of the homogeneous algorithm for large-scale convex optimization. *Comput. Optim. Appl.* **1998**, *10*, 243–269.
25. Andersen, E.D.; Ye, Y. On a homogeneous algorithm for the monotone complementarity problem. *Math. Program.* **1999**, *84*, 375–399.
26. Domahidi, A.; Chu, E.; Boyd, S. ECOS: An SOCP solver for embedded systems. In Proceedings of the European Control Conference (ECC), Zurich, Switzerland, 17–19 July 2013; pp. 3071–3076.
27. O’Donoghue, B.; Chu, E.; Parikh, N.; Boyd, S. Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding. *J. Optim. Theory Appl.* **2016**, *169*, 1042–1068.
28. O’Donoghue, B.; Chu, E.; Parikh, N.; Boyd, S. SCS: Splitting Conic Solver, Version 1.2.7. Available online: <https://github.com/cvxgrp/scs> (accessed on 6 October 2017).
29. Grant, M.; Boyd, S. CVX: Matlab Software for Disciplined Convex Programming, Version 2.1. Available online: <http://cvxr.com/cvx> (accessed on 6 October 2017).



© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).