Maximizing a Sum of Sigmoids

Madeleine Udell Stephen Boyd

May 15, 2015

Abstract

The problem of maximizing a sum of sigmoidal functions over a convex constraint set arises in many application areas. This objective captures the idea of decreasing marginal returns to investment, and has applications in mathematical marketing, network bandwidth allocation, revenue optimization, optimal bidding, and lottery design. We define the sigmoidal programming problem (SP) and show how it arises in each of these application areas. We show that the general problem is NP-hard, and propose an approximation algorithm (using a branch and bound method) to find a globally optimal approximate solution to the problem. We show that this algorithm finds approximate solutions very quickly on problems of interest: in fact, for problems with few constraints, it frequently finds a solution within an acceptable error margin by solving a single convex optimization problem. To illustrate the power of this approach, we compute the positions which might have allowed the candidates in the 2008 United States presidential election to maximize their vote shares.

1 Introduction

1.1 Overview

The ability to efficiently solve large classes of convex optimization problems has enabled many of the greatest advances in operations research, machine learning, and control. By posing problems as convex programs, researchers and practitioners are able to take advantage of standard and scalable solvers which allow them to quickly find a global solution to their problems. When confronted with a non-convex problem, researchers may be tempted to give up hope of finding a global solution, and instead rely on heuristics and local optimization procedures. However, the quality of the solution obtained in this manner is generally unknown.

In this paper, we define a class of non-convex, NP-hard problems which we call *sigmoidal programs*, and describe an algorithm to find provably optimal global solutions. A sigmoidal program (SP) resembles a convex program, but allows a controlled deviation from convexity in the objective function. The framework that we present for sigmoidal programming is general enough to capture a wide class of objective functions, and any convex constraint set.

Our algorithm for sigmoidal programming relies on the well-known branch and bound method for non-convex optimization (Lawler and Wood 1966, Balas 1968). We compute upper and lower bounds for the sigmoidal programming problem by relaxing it to a tractable convex program. These upper and lower bounds are used as the basis of a recursion that eventually converges to the global solution to the problem. Sigmoidal programming derives its speedy performance in practice from the ease with which the convex relaxation may be computed. The time required to compute a solution using our proposed algorithm is a small multiple of the time required to solve a linear program, for problems that are "almost" convex programs or for problems with a small number of constraints.

The main contributions of this paper are the identification of SP as a broad problem class with numerous applications; a method for constructing concave envelopes of sigmoidal functions given a first order oracle function interface, which makes the branch and bound approach computationally feasible; and a self-contained proof of convergence. The authors have also released an open source software package, `SigmoidalProgramming`, which implements the algorithm described here.

1.2 Outline

We first define the class of functions and problems that can be optimized using sigmoidal programming. We move on to applications in §3, and give a few examples of domains in which sigmoidal programming may be useful, and we discuss related work in §4. In §5 we prove the problem class is NP-hard, and give some results on approximability of SP. We describe our method for solving sigmoidal programming problems in §6. In §7, we describe the software package `SigmoidalProgramming` and report the results of a few numerical experiments using this software. In Appendix A, we prove that our method always converges to the optimal solution in (worst-case) exponential time, and in Appendix B we

provide a stronger complexity result.

2 Problem definition

In this paper, we consider the *sigmoidal programming problem*

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} f_i(x_i) \\ \text{subject to} & x \in \mathcal{C}, \end{array} \tag{1}$$

where $f_i(x) : [l, u] \to \mathbf{R}$ is a sigmoidal function for each i, and the variable $x \in \mathbf{R}^n$ is constrained to lie in a nonempty bounded closed convex set $\mathcal{C}$.

A Lipshitz continuous function $f : [l, u] \to \mathbf{R}$ is defined to be *sigmoidal* if it is either convex, concave, or convex for $x \leq z \in [l, u]$ and concave for $x \geq z$ for some parameter $z \in \mathbf{R}$ (see Figure 1).

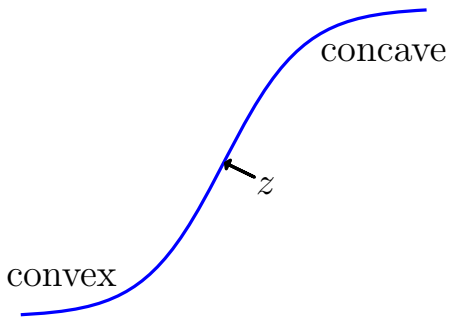

Figure 1: A sigmoidal function with inflection point at z.

2.1 Sigmoidal functions

Sigmoidal functions arise in a number of modeling contexts. We note first that all convex, concave, and affine functions are sigmoidal, according to our definition. The class also includes those functions whose graphs are "s-shaped", including the logistic function $\text{logistic}(x) = \exp(x)/(1 + \exp(x))$, and the error function $\text{erf}(x) = 2/\sqrt{\pi} \int_0^x \exp(-t^2)dt$. More generally, the cumulative distribution function (CDF) of any bounded quasi-concave probability distribution is sigmoidal. See Figure 2 for a few examples of sigmoidal functions.

Statistical models. Sigmoidal functions arise in the guise of CDFs in machine learning models for binary events. For example, if the probability of winning an auction or an election is fit using a logit or probit model, then that model gives the probability of winning as a sigmoidal function of the control parameters used to fit the model, such as money or time invested. The problem of winning a number of auctions or votes from separately modelled groups then becomes a sigmoidal programming problem.

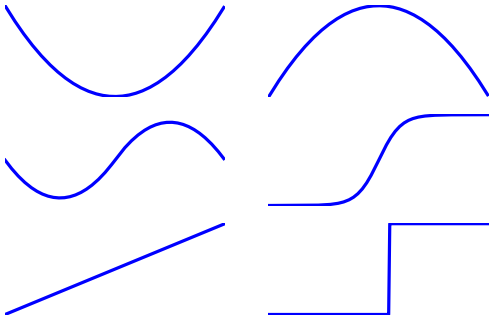

Figure 2: A few examples of sigmoidal functions.

Utility functions. In economics, the idea that curvature of the utility function might change sign dates back at least to Friedman and Savage (1948), who considered utility functions that were concave at low and high income levels, and convex in between (see Figure 3). They argued that the eponymous Friedman-Savage utility function might explain the willingness of a person to buy insurance for large losses (concave utility for losses) while also playing the lottery (convex utility for medium gains). The concavity of the utility function for extremely high gains was used to explain why lottery prizes are typically divided into many prizes of roughly similar size, as though to extend the income offered by the prize only to the upper limit of the convex portion of the utility curve. The Friedman-Savage utility function can be written as the sum of two sigmoids using the decomposition given below in equation (2). More recently, prospect theory has also posited sigmoidal utility functions for similar reasons (Kahneman and Tversky 1979, Tversky and Kahneman 1992). Hence the classical problem of societal utility maximization may be viewed as a sigmoidal programming problem.

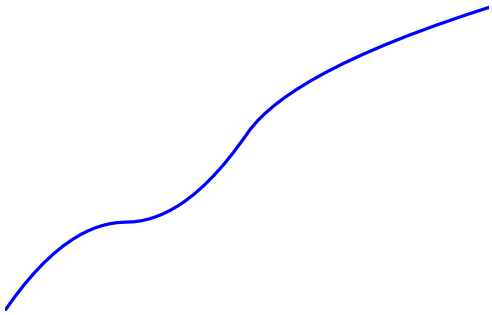

Figure 3: A Friedman-Savage utility function.

Step functions. Sigmoidal functions may also be used to approximate a step function to any arbitrary accuracy: for example, the *admittance* function

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x/\epsilon & 0 < x < \epsilon \\ 1 & x \geq \epsilon \end{cases},$$

which approximates a step function arbitrarily well as $\epsilon \to 0$, is sigmoidal for any $\epsilon > 0$. Sigmoidal functions that approximate a step function can be used to describe the utility from goods that are only useful in sufficient quantities, such as communications bandwidth for videoconferencing, or votes in a first-past-the-post election. Thus the problems of network bandwidth allocation and of winning elections in an electoral college system fit naturally in the sigmoidal programming paradigm.

Economies of scale. Sigmoidal functions aptly describe the profits of a firm that enjoys economies of scale as it increases to a moderate size and diseconomies of scale when it grows excessively large.

Everything else. It is worthwhile noting that while our definition of a sigmoidal function allows for only one inflection point, there is also a simple reduction of a known-inflection-point problem to a sigmoidal programming problem. Any function whose k inflection points are all known may be written as the sum of $k-1$ of sigmoidal functions.

For example, if $f : [l, u] \to \mathbf{R}$ is convex on $[l, z_1]$, concave on $[z_1, z_2]$, and convex on $[z_2, u]$, then f may be written as

$$f(x) = f_1(x) + f_2(x) \tag{2}$$

where f_1 and f_2 are both sigmoidal, *i.e.*,

$$\begin{aligned} f_1(x) &= \begin{cases} f(x) - 1/2 f'(z_2)(x - z_2) & x \leq z_2 \\ f(z_2) + 1/2 f'(z_2)(x - z_2) & x > z_2 \end{cases} \\ f_2(x) &= \begin{cases} 1/2 f'(z_2)(x - z_2) & x \leq z_2 \\ f(x) - f(z_2) - 1/2 f'(z_2)(x - z_2) & x > z_2 \end{cases} \end{aligned}$$

(see Figure 4). (Note that we must add a constraint requiring the arguments of f_1 and f_2 to be equal in order to fit the standard form SP.) It is also easy to search for inflection points z numerically using bisection.

Hence an algorithm for sigmoidal programming is effectively an algorithm for optimizing sums of functions with known curvature. Furthermore, sums of sigmoidal functions of the form

$$\sum_{i=1,\ldots,n} f_i(a_i y + b_i)$$

are dense in the space of continuous functions on a bounded domain $\mathcal{D} \subset \mathbf{R}^m$ (Cybenko 1989). Hence we can approximate any continuous function arbitrarily well by a suitably large linear combination of sigmoidal functions $f_i(x_i)$ if we add to the problem the affine constraint $x = Ay + b$ for some $y \in \mathbf{R}^m$.

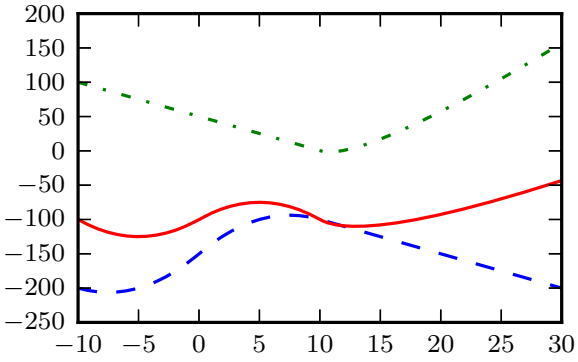

Figure 4: Decomposition of a function with two inflection points into two sigmoidal functions. The function f (solid line) is the sum of f_1 (dashed line) and f_2 (dot-dashed line).

3 Applications

In this section, we give a few applications of sigmoidal programming. The paradigm of sigmoidal programming is particularly well suited to solve *allocation problems*, in which a decision maker seeks to allocate a number of scarce resources between several objectives. Allocation problems arise naturally as a consequence of a decision maker's desire to maximize the positive outcomes of each of a number of different models, subject to resource constraints.

3.1 Machine learning models

For example, suppose an agent wishes to target each of n groups, with populations $p_i > 0$, $i = 1, \ldots, n$. The agent might be an advertising agency seeking to win market share, a political campaign seeking to win votes, a public health agency seeking to prevent disease, or a law enforcement agency seeking to detect criminal activity. The agent has access to a model for the efficacy of his actions on each group that depends on the quantity of a number m of scarce resources allocated to each group. This model gives the expected proportion $f_i(w_i^T y_i)$ of the group i for which the agent will be successful as a sigmoidal function f_i of a linear combination of the resources $y_i \in \mathbf{R}^m$ allocated to that group, where $w_i \in \mathbf{R}^m$ and $f_i : \mathbf{R} \to \mathbf{R}$ are parameters of the model, which we assume are given, and characterize the expected reaction of each market segment to the agent's actions. We may have a constraint on the total amount of each resource allocated,

$$\sum_{i=1}^{n} y_i \leq Y,$$

for some $Y \in \mathbf{R}^m$, and also a constraint on how much of each resource may be used for each group,

$$y^{\min} \leq y_i \leq y^{\max}, \quad i = 1, \ldots, n,$$

where $y^{\min}, y^{\max} \in \mathbf{R}^m$. The expected population that will be swayed by the agent's actions, summed over all groups, is

$$\sum_{i=1}^{n} p_i f_i(w_i^T y_i).$$

The problem is to choose y so as to maximize this quantity.

We can write this problem as a standard form sigmoidal programming problem using an auxiliary variable x whose ith component represents the linear combination of resources $w_i^T y_i$. The problem can be written as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n p_i f_i(x_i) \\ \text{subject to} & \sum_{i=1}^n y_i \leq Y \\ & y^{\min} \leq y_i \leq y^{\max}, \quad i = 1, \ldots, n \\ & x_i = w_i^T y_i, \quad i = 1, \ldots, n. \end{array}$$

3.2 Cumulative distribution functions

Sigmoidal functions can arise as the cumulative distribution function of any quasi-concave probability distribution. We show in this section how to cast an optimal bidding problem as a sigmoidal programming problem.[1]

A bidder at an auction has a budget of B with which to bid on n different goods. The bidder privately believes that each good has a value v_i, $i = 1, \ldots, n$, and models the likelihood of winning the item as a sigmoidal function f_i of the bid b_i, $i = 1, \ldots, n$. The value derived by the bidder from a given bid is the expected profit from that bid, $(v_i - b_i) f_i(b_i)$. The bidder will not tolerate a negative expected profit, so we restrict our attention to bids $b_i \leq v_i$, $i = 1, \ldots, n$.

The problem is to maximize the total expected profit,

$$\sum_{i=1}^{n} (v_i - b_i) f_i(b_i),$$

subject to the limits on the bid values, $\sum_{i=1}^n b_i \leq B$ and $b_i \leq v_i$, $i = 1, \ldots, n$.

It is a simple exercise in univariate calculus to show that

$$(v_i - b_i) f_i(b_i)$$

is sigmoidal on the interval $(-\infty, v_i)$ if f_i is a sigmoidal CDF for every $i = 1, \ldots, n$.

Lottery design. We note that the opposite problem may also be of interest: that of designing a lottery or auction system that maximizes profit to the proprietor. Friedman and Savage (1948) argue that the curvature of the utility function implies that there is a unique

[1] We thank AJ Minich for developing this formulation of the optimal bidding problem in his class project for EE364b (Spring, 2011) at Stanford.

prize amount that lotteries ought to offer to maximize profit. The fact that lotteries often split their top prize into two or three equally sized prizes is evidence for their conjecture. Using sigmoidal programming, we can quantitatively test this theory. Suppose that all people have the same utility curve $f(x)$ as a function of income x. (This assumption makes no difference to the argument, but simplifies the notation.) The expected utility a person derives from a lottery ticket with prizes $x_i \geq 0$, $i = 1, \ldots, n$, and a cost per ticket $c > 0$ is

$$\mathbf{E}[f(x_i - c)] = \frac{1}{n} \sum_{i=1}^{n} f(x_i - c).$$

Participants will be willing to buy tickets for the lottery so long as the expected utility gain of participating is positive. The profit of the proprietor of the lottery is given by $nc - \sum_{i=1}^{n} x_i$. Hence a given profit P is feasible if the optimal value of the problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} f(x_i - c) \\ \text{subject to} & nc - \sum_{i=1}^{n} x_i \geq P \\ & x_i \geq 0, \quad i = 1, \ldots, n \\ & c > 0 \end{array}$$

is greater than zero. This problem can be transformed into a sigmoidal problem by decomposing the utility function into sigmoidal functions and introducing auxiliary variables, as was shown in §2.1.

The proprietor of the lottery maximizes his profit by finding the largest value of P such that the optimal value of (3.2) is positive, which can be computed by solving a sequence of sigmoidal programming problems to find a maximal P. Conversely, economists might use sigmoidal programming to infer the shape of lottery players' utility functions from a schedule of prizes.

3.3 Economies of scale

Sigmoidal functions also arise naturally in problems involving economies and diseconomies of scale. We give an example from revenue optimization.

A firm expects to enjoy economies of scale in the production of each of n goods, leading to increasing marginal returns as the amount of each good produced increases. However, the total market for each finished product is finite, and as the quantity of the good produced increases, the marginal return for each product will diminish or even become negative. The characteristic shape of the revenue curve is given by a function f_i for each good $i = 1, \ldots, n$, which is concave for large production volume, when the beneficial economies of scale are outweighed by the negative price pressure due to market saturation. The total expected revenue from product i if a quantity y_i of the good is produced is $f_i(y_i)$.

The quantity of each good produced is limited by the availability of each of m inputs, which might include money, labor, or raw materials. The plant has access to a quantity z_j of each input $j = 1, \ldots, m$, and can choose to allocate α_{ij} of each input j to the production of each good $i = 1, \ldots, n$, so long as $\sum_{i=1}^{m} \alpha_{ij} \leq z_j$. The firm requires γ_{ij} of input j for every

input $j = 1, \ldots, m$ to produce each unit of output i, so the total production y_i is always controlled by the limiting input,

$$y_i \leq \min_j \gamma_{ij}\alpha_{ij} \quad i = 1, \ldots, n.$$

The firm may also be subject to sector constraints limiting investment in each of a number of sectors either in absolute size or as a proportion of the size of the firm. These constraints can be written, respectively, as

$$Ay \leq b$$

or

$$Ay \leq \delta \mathbf{1}\mathbf{1}^T y,$$

where A is a matrix mapping outputs y into sectors, b gives absolute constraints on the size of each sector, δ is the maximum proportion of the total output that may be concentrated into any single sector, and $\mathbf{1}$ is the vector of all ones.

The total revenue of the firm can be written as

$$R = \sum_{i=1}^{n} f_i(y_i).$$

The problem is to choose y (subject to the input and sector constraints) so as to maximize R.

3.4 Network utility maximization

Sigmoidal programming has previously received attention as a framework for network utility maximization (NUM). Fazel and Chiang (2005) consider the problem of maximizing the utility of a set of flows through a network $G = (V, E)$, where the utility of a flow is a sigmoidal function of the volume of the flow, and each edge $e \in E$ is shared by many flows $i \in S(e)$ in the network. Here, we let x_i denote the volume of each flow $i = 1, \ldots, n$ and $f_i(x_i)$ the utility of that flow, while $c(e)$ gives the capacity of edge e. Fazel and Chiang consider only utilities that are expressible as polynomials f_i, which are required by their solution method. Using sigmoidal programming, one can solve NUM problems with polynomial utilities, but we can also solve problems in which utilities take other (sigmoidal) forms. We might take the utility to be an *admittance* function,

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x/\epsilon & 0 < x < \epsilon \\ 1 & x \geq \epsilon. \end{cases}$$

For example, the utility of network bandwidth to be used for videoconferencing or other realtime applications is nearly zero until a certain flow rate can be guaranteed, and saturates when the application is able to send data at the same rate as it is produced.

The NUM problem is to maximize the total utility of the flows $\sum_{i=1}^n f_i(x_i)$ subject to the bandwidth constraints $\sum_{i \in S(e)} x_i \leq c(e)$. Define the edge incidence matrix $A \in \mathbf{R}^{|E| \times n}$ mapping flows onto edges, with entries a_{ei}, $i = 1, \ldots, n$, $e \in E$. An entry $a_{ei} = 1$ if $i \in S(e)$, *i.e.*, if flow i uses edge e, and 0 otherwise. We write the NUM problem as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n f_i(x_i) \\ \text{subject to} & Ax \leq c \\ & x \geq 0. \end{array}$$

4 Related work

While our treatment of sigmoidal programming problems as a distinct problem class is new, our methods have deep historical roots. Below, we review some of the previous work for readers interested in a more thorough treatment of these topics.

Branch and bound. The branch and bound method for solving non-convex optimization problems has been well-known since the 1960s; see, for example, Balas (1968) for a brief overview of the branch and bound method applied to problems with discrete optimization variable, or Lawler and Wood (1966) for a more detailed review of the general method with continuous and discrete examples.

A number of authors working before the advent of fast routines for convex optimization examined the possibility of using branch and bound techniques to solve nonconvex separable problems (McCormick 1976, Falk and Soland 1969). As in this paper, the authors suggested solving convex subproblems based on convex envelopes as part of a larger branch and bound routine; but these papers did not give computationally efficient routines for computing the convex envelopes of the general univariate functions f_i they considered.

This deficiency was remedied by later works in the global optimization community: see, eg, Horst et al (2000) for a general introduction and Tawarmalani and Sahinidis (2002) for a more sophisticated treatment. For example, Tawarmalani and Sahinidis (2002) discuss an efficient procedure similar to the one presented here in §6 for computing the concave envelope of sigmoidal functions (there called "convexoconcave").

A number of global optimization solvers have been developed based on these ideas to solve complex, but usually small-scale, nonlinear (nonconvex) optimization problems (NLPs). These solvers generally rely on the two ideas mentioned above — convex envelopes and the branch and bound method — along with many others (such as branch and cut, constraint propagation, interval analysis, duality, etc.). A number of tuning parameters govern the relative frequency with which each of these heuristics is used. As a consequence, these solvers are complex engineered objects that are in general expensive to develop and to maintain. Many (including BARON (Sahinidis 2014, Tawarmalani and Sahinidis 2005) and ANTIGONE (Misener and Floudas 2014)) are offered as commercial software with a hefty license fee even for academic use; their open source competitors (such as Couenne (Belotti 2009) and SCIP (Achterberg 2009)) run substantially slower on (and fail to solve many problems in) a reference test set of global optimization problems (Sahinidis 2014).

Sigmoidal programming. The sigmoidal programming algorithm we present in this paper, and the associated software package, fill a complementary niche. They are targeted at a problem with one particularly simple, but common, form; so the algorithm (presented in §6) is easy to understand, and the implementation, `SigmoidalProgramming`, is short (about 300 lines of code), and easy to understand or to extend. To our knowledge, this paper presents the first focused treatment of sigmoidal programming as a distinct (and particularly simple) problem class.

The simplicity of sigmoidal programs has further advantages. First, it allows us to make guarantees about the maximal error at each branch and bound node by examining the structure of the sigmoidal program (§5). In fact, for large problems, the algorithm frequently find a solution within an acceptable error margin by solving a single convex optimization problem (§7).

Second, our algorithm requires minimal information about the sigmoidal functions in the sum: it merely relies on an oracle to compute function values and (super)gradients. In particular, it does not require access to the algebraic form of the functions to compute their concave envelopes.

Problems of intermediate complexity (between sigmoidal programs and general NLPs) may be tackled with solvers of intermediate complexity. For example, D'Ambrosio et al (2012) give a framework for solving mixed integer nonlinear programs with separable nonconvexity, which generalize sigmoidal programs by allowing sums of univariate nonconvex functions (which may have multiple regions of concavity and convexity) to appear in the constraints as well as in the objective of the problem.

Convex optimization. In this paper, we treat convex optimization methods as an oracle for solving the subproblems that emerge in the iterations of the branch and bound method. That is, we compute the complexity of our algorithm in terms of the number of calls to a convex solver. The reader unfamiliar with convex optimization might consult Boyd and Vandenberghe (2004) for a comprehensive treatment of these methods and their applications. Linear programming solvers suffice as oracles for the applications we describe, all of which have only linear constraints; however, our convergence results and the algorithm presented encompass more general convex constraints, such as second order cone (SOCP) or semidefinite (SDP) constraints. Many fast solvers for LP, SOCP, and SDP are freely available in a variety of programming languages. See, for example, cvxopt (Andersen et al 2013), SeDuMi (Sturm 1999), SDPT3 (Toh et al 1999), and GLPK (Makhorin 2006).

Bounds without branches. Other authors have previously considered global optimization algorithms not based on the branch and bound method for certain subclasses of sigmoidal programs. For example, motivated by an application to network utility maximization (NUM), Fazel and Chiang (2005) explore an algorithm to compute global bounds on the optimal value of sigmoidal programs with polynomially representable objective functions. They compute an upper bound on the optimal value using a sum of squares decomposition (Parrilo 2003), and a lower bound via a method of moments relaxation (Lasserre 2001, Henrion

and Lasserre 2005), which are both found by solving an SDP. The quality of the bounds is controlled by the degree of polynomial allowed in the SDP; they find a small degree generally gives fine accuracy. However, even for small degrees, the complexity of these SDPs grow so quickly in the number of variables in the problem that all but the simplest problems are prohibitively expensive to solve. For example, the largest numerical example in (Fazel and Chiang 2005) has only 9 variables and 7 constraints.

Difference of convex programmming. Difference of convex (DC) programming (Horst et al 2000, Lipp and Boyd 2014) can also be used to solve sigmoidal problems. The DC technique allows global optimization of a difference of (multivariate) convex functions. It is easy to see that sigmoidal functions can be written in this form (using a decomposition technique similar to equation (2)). However, the concave envelope of the convex function over the region will never be tighter than the concave envelope of the sigmoidal function (see §6.1). Hence the algorithm is unlikely to converge more quickly.

5 Complexity

Hardness of sigmoidal programming. It is easy to see that sigmoidal programing is NP-hard. For example, one can reduce integer linear programming,

$$\begin{array}{ll} \text{find} & x \\ \text{subject to} & Ax = b \\ & x \in \{0,1\}^n, \end{array}$$

which is known to be NP-hard (Karp 1972), to sigmoidal programming:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n g(x_i) = x_i(x_i - 1) \\ \text{subject to} & Ax = b \\ & 0 \leq x_i \leq 1, \quad i = 1, \ldots, n, \end{array}$$

where $g(x)$ is a function chosen to enforce a penalty on non-integral solutions, *e.g.*, $g(x) = x(x-1)$. Then the solution to the sigmoidal program is 0 if and only if there is an integral solution to $Ax = b$.

This reduction provides some insight into which parameters of the problem result in its difficulty, since the number of variables and constraints in the ILP map exactly onto the number of variables and constraints in the SP.

In Appendix B, we prove the following stronger result, using a reduction to maximum independent set (MIS) (Berman and Schnitger 1992, Hastad 1996, Beigel 1999).

Theorem 1. *It is NP-hard to compute an ρ-approximate solution to the sigmoidal programming problem*

$$\begin{array}{ll} \textit{maximize} & \sum_{i=1}^n f_i(x_i) \\ \textit{subject to} & x \in \mathcal{C}, \end{array}$$

where $f_i(x)$ is sigmoidal $i = 1, \ldots, n$ and $\rho > n^{1-\epsilon}$, for any $\epsilon > 0$.

Approximation guarantees. On the other hand, it is well know that special cases of NP-hard problems are often significantly easier than the general case. Here, we show that sigmoidal programming works extremely well for problems with a small number of linear constraints. Suppose that

$$\mathcal{C} = \{x \mid Ax \leq b,\ Gx = h\},$$

with $A \in \mathbf{R}^{m_1 \times n}$, so there are m_1 linear inequality constraints, and $G \in \mathbf{R}^{m_2 \times n}$, so there are m_2 linear equality constraints. Let $m = m_1 + m_2$ be the total number of constraints.

It is possible to obtain an approximate solution for SP in polynomial time whose quality depends only on the number of constraints and the non-convexity of the functions f_i, and not on the dimension n of the problem, when $m < n$. More precisely, let $\hat{f}(x)$ be the *concave envelope* of the function f, which is defined as the pointwise infimum over all concave functions g that are greater than or equal to f on its domain. Define the *nonconvexity* $\rho(f)$ of a function $f : S \to \mathbf{R}$ to be

$$\rho(f) = \sup_x (\hat{f}(x) - f(x)).$$

Consider an (approximate) solution to SP obtained by solving a convex relaxation of the original problem, in which each function f is replaced by its concave envelope $\hat{f}$. In a related paper (Udell and Boyd 2014), the authors show that there exists a solution to this relaxed problem for which the approximation error is bounded by

$$\sum_{i=1}^{\min(m,n)} \rho_{[i]},$$

where we define $\rho_{[i]}$ to be the ith largest of the nonconvexities $\rho(f_1), \ldots, \rho(f_n)$. Hence for problems that are close to convex (*i.e.*, $\rho_{[1]}$ is small), or with very few constraints or variables (*i.e.*, m or n is small), we can expect that we can find a good approximate solution in a very small number of iterations. We take advantage of this property in the algorithm presented in §6.

The ease of solving sigmoidal programming problems with a small number of constraints makes this approach particularly well suited to solve allocation problems with a small number of resources. Indeed, we will see in §7 that all of the numerical examples we consider (including a problem with 10,000 variables) can be solved to very good accuracy in a few tens of iterations.

6 Method

The algorithm we present for sigmoidal programming uses a concave approximation to the problem in order to bound the function values and to locate the local maxima that are high enough to warrant consideration. In the worst case we may solve exponentially many convex optimization problems, but frequently we find bounds that are sufficiently tight after solving only a small number of concave subproblems. Furthermore, we may terminate the algorithm

at any time to obtain upper and lower bounds on the possible optimal value, along with a feasible point that realizes the lower bound.

A sketch of the branch and bound algorithm we use is given as Algorithm 1. The **update** ensures that LB is always the best global lower bound, that UB is the best global upper bound, and that $\mathcal{Q}$ remains a partition of Q_0. Within this algorithmic framework, we specialize to the case of sigmoidal programs by choosing how to **compute** upper and lower bounds, and how to **split** each rectangle:

- We **compute** upper and lower bounds by solving the convex problem (4).
- We **split** Q along the coordinate i for which the upper and lower bound disagree maximally, at $\min(x_i^\star(Q), z_i)$, where $x^\star(Q)$ solves problem (4).

We describe the steps in detail below, and give intuition for correctness and convergence. A formal proof of convergence is deferred to Appendix A.

Algorithm 1 Branch and bound

given initial rectangle Q_0, tolerance ϵ
compute lower bound $L(Q_0)$ and upper bound $U(Q_0)$ on Q_0
initialize global lower bound LB= $L(Q_0)$, global upper bound UB= $U(Q_0)$, partition $\mathcal{Q} = \{Q_0\}$
while UB - LB $> \epsilon$ **do**
  select $Q \in \mathcal{Q}$ with highest upper bound $U(Q)$
  update global upper bound UB $\leftarrow U(Q)$
  split Q into left and right subrectangles Q_L and Q_R
  compute lower bound $L(Q_L)$ and upper bound $U(Q_L)$ on Q_L
  compute lower bound $L(Q_R)$ and upper bound $U(Q_R)$ on Q_R
  update global lower bound LB $\leftarrow \max(\text{LB}, L(Q_L), L(Q_R))$
  update partition $\mathcal{Q} = \mathcal{Q} \cup \{Q_L, Q_R\} \setminus \{Q\}$
end while

Branch and bound. The branch and bound method (Lawler and Wood 1966, Balas 1968, Balakrishnan et al 1991) is a recursive procedure for finding the global solution to an optimization problem restricted to a bounded rectangle Q_{init}. The method works by partitioning the rectangle Q_{init} into smaller rectangles $Q \in \mathcal{Q}$, and computing upper and lower bounds on the value of the optimization problem restricted to those small regions. Denote by $p^\star(Q)$ the optimal value of the problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n f_i(x_i) \\ \text{subject to} & x \in \mathcal{C} \cap Q \end{array} \tag{3}$$

for any rectangle $Q \in \mathcal{Q}$. The global optimum must be contained in one of these rectangles, so if $U(Q)$ and $L(Q)$ denote the upper and lower bounds on $p^\star(Q)$,

$$L(Q) \leq p^\star(Q) \leq U(Q),$$

then

$$\max_{Q\in\mathcal{Q}} L(Q) \leq p^\star(Q_{\text{init}}) \leq \max_{Q\in\mathcal{Q}} U(Q).$$

The method relies on the fact that it is easier to compute tight bounds on the function value over small regions than over large ones. Hence by *branching* (dividing promising regions into smaller subregions), the algorithm obtains global bounds on the value of the solution that become arbitrarily tight.

Bound. In sigmoidal programming, we quickly compute a lower and upper bound on $p^\star(Q)$ by solving a convex optimization problem. Let $Q = I_1 \times \cdots \times I_n$ be the Cartesian product of the intervals $I_1, \ldots, I_n$. For each function f_i, $i = 1, \ldots, n$ in the sum, suppose that we have a concave upper bound $\hat{f}_i$ on the value of f_i over the interval I_i,

$$\hat{f}_i(x) \geq f_i(x), \quad x \in I_i.$$

Let $x^\star(Q)$ be the solution to the convex problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n \hat{f}_i(x_i) \\ \text{subject to} & x \in \mathcal{C} \cap Q. \end{array} \tag{4}$$

This problem has the same set of feasible points as problem (3), and for each feasible point its objective value is at least as large. Hence we obtain an upper bound $U(Q)$ on the optimal value $p^\star(Q)$:

$$U(Q) = \sum_{i=1}^n \hat{f}_i(x_i^\star(Q)) \geq p^\star(Q). \tag{5}$$

(Note that problem (4) could be infeasible, in which case we know that the solution to the original problem cannot lie in Q.)

To construct a lower bound, note that the value of the objective at any feasible point gives a lower bound on $p^\star(Q)$, and so in particular,

$$L(Q) = \sum_{i=1}^n f_i(x_i^\star(Q)) \leq p^\star(Q). \tag{6}$$

Concave envelope. If we choose $\hat{f}_i$ to be concave, then subproblem (4) is a convex optimization problem, and can be solved efficiently (Boyd and Vandenberghe 2004). The tightest concave approximation to f is obtained by choosing $\hat{f}$ to be the *concave envelope* of the function f, which is defined as the pointwise infimum over all concave functions g that are greater than or equal to f. For a sigmoidal function, the concave envelope is particularly easy to calculate. We can write $\hat{f}$ of f piecewise as

$$\hat{f}(x) = \begin{cases} f(l) + \frac{f(w)-f(l)}{w-l}(x-l) & l \leq x \leq w \\ f(x) & w \leq x \leq u, \end{cases}$$

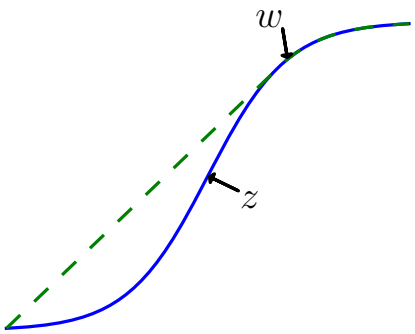

Figure 5: The concave envelope $\hat{f}$ (dashed line) of the logistic function f (solid line).

for some $w > z$ (see Figure 5). The point w can easily be found by bisection: if $x < w$, then the line from $(l, f(l)$ to $(x, f(x)$ crosses the graph of f at x (from lying above the graph to lying below); if $x > w$, it crosses in the opposite direction.

When we use the concave envelope to compute the bounds (5) and (6), and the constraint set $\mathcal{C}$ consists only of m linear equalities and inequalities, Theorem 2 from (Udell and Boyd 2014) guarantees that these bounds satisfy

$$U(Q) - L(Q) \leq \sum_{i=1}^{\min(m,n)} \rho_{[i]},$$

where

$$\rho(f) = \sup_{x \in Q}(\hat{f}(x) - f(x))$$

and $\rho_{[i]}$ is defined to be the ith largest of the nonconvexities $\rho(f_1), \ldots, \rho(f_n)$. As the algorithm proceeds, the size of the rectangles decreases, and so $U(Q) - L(Q)$ also decreases.

Piecewise linear upper bound. When all the constraints are linear, it may be convenient to construct a piecewise linear concave upper bound on the function f, and to maximize this piecewise linear upper bound instead of the concave envelope (see Figure 6). In this case, if $\mathcal{C}$ is a polyhedron, the computation of upper and lower bounds on $p^\star(Q)$ reduces to a linear programming problem (Boyd and Vandenberghe 2004).

If f is differentiable, a piecewise linear concave upper bound, $\bar{f}$, may be constructed as

$$\bar{f}(y) = \inf_{x \in S'} f(x) + f'(x)(y - x)$$

where $S' = \{w\} \cup S$ for any $S \subseteq [w, \infty)$. Since the maximum error $\hat{f}(x) - \bar{f}(x)$ in the approximation must occur where adjacent lines intersect, the points of intersection may be

added to the set S' until the maximum error is less than ϵ, producing an upper bound that satisfies $\sup_x \hat{f}(x) - \bar{f}(x) \leq \epsilon$.

If f is not differentiable, define $f'(w) = \frac{f(w)-f(l)}{w-l}$, and let $f'(x)$ be any supergradient of f at x for $x > w$. (A supergradient g of f at x satisfies $f(y) \leq f(x) + g(y-x)$ for any $y \in [z, \infty)$.)

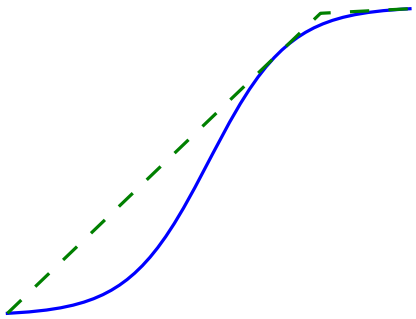

Figure 6: A piecewise linear approximation (dashed line) to the concave envelope of the (solid line) logistic function.

This approach to solving subproblem (4) has the advantage that it requires minimal information about the functions f_i: we need only know

1. an oracle for computing the value of the function at a point,
2. an oracle for computing a supergradient of the function at a point, and
3. the value z_i (so that we can compute w_i).

(As mentioned above, z_i can also be easily computed by bisection as a preprocessing step.) In particular, this means that a solver using this approach to solve subproblem (4) need not know the full functional form of the functions f_i. Hence users can add their own custom sigmoidal functions f_i with relative ease.

Cutting plane upper bound. A cutting plane method may also be used to solve subproblem (4). This approach is similar to the method described above for computing a piecewise linear concave upper bound, but has the advantage that not all pieces of the piecewise linear upper bound need be constructed.

For each function f_i in the sum, introduce an epigraph variable t_i. Instead of maximizing $\sum_{i=1}^n \hat{f}_i(x_i)$, we shall maximize $\sum_{i=1}^n t_i$, and add as many linear constraints as are necessary to ensure that $\bar{t}_i \leq \hat{f}_i(\bar{x}_i)$ at the solution $(\bar{x}, \bar{t})$. For each $i = 1, \ldots n$, we begin with just two constraints on each epigraph variable t:

$$t \leq f_i'(w_i)(x - w_i) + f_i(w_i) \qquad t_i \leq f_i'(u_i)(x - u_i) + f_i(u_i),$$

and add more as needed. (This notation assumes that f_i is differentiable. If it is not, then define $f_i'(w_i) = \frac{f_i(w_i)-f_i(l_i)}{w_i-l_i}$, and let $f_i'(x)$ be any supergradient of f_i at x for $x > w_i$. We will not need to invoke $f_i'(x)$ for any $x < w_i$.)

More formally, start by setting $C_i = \{w_i, u_i\}$, and solve the problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n t_i \\ \text{subject to} & t_i \leq f_i'(y)(x_i - y) + f_i(y), \quad y \in C_i. \end{array} \tag{7}$$

Now using the solution $(\bar{x}, \bar{t})$ to this problem, for each $i = 1, \ldots n$, check if $\bar{t}_i > \hat{f}_i(\bar{x}_i)$ and if so, update $C_i = C_i \cup \{\bar{x}_i\}$. If any of the sets C_i have changed, solve problem (7) again, and repeat until the sets C_i do not change.

Alternatively, we may choose to add a new point to C_i only if $\bar{t}_i > \hat{f}_i(\bar{x}_i) + \epsilon/n$ for some tolerance ϵ. In this case, the algorithm will terminate with

$$\sum_{i=1}^n \hat{f}_i(\bar{x}_i) \leq \sum_{i=1}^n \bar{t}_i \leq \sum_{i=1}^n \hat{f}_i(\bar{x}_i) + \epsilon,$$

giving an upper bound $\sum_{i=1}^n \bar{t}_i$ and lower bound $\sum_{i=1}^n \hat{f}_i(\bar{x}_i)$ on the optimal value of problem (4) that differ by no more than ϵ.

This cutting plane approach can be made particularly fast by using the previous solution $(\bar{x}, \bar{t})$ as a hot-start for problem (7) when new points are added to the sets C_i. This is the approach adopted to solve the subproblems (4) in the `SigmoidalProgramming` implementation of this method. Note that the cutting plane approach also requires the same oracle-style interface to the functions f_i as the piecewise linear upper bound approach.

Branch. The concave envelope of f over the interval (l, u) is equal to f at l and u, and in general will lie closer to f on small intervals, or on intervals over which f is less strongly convex (closer to linear). These properties allow us to find tighter bounds on the optimal value of the problem over a smaller region than over a larger one, with the size required to find a bound of a given tightness controlled by the convexity of the function. Hence, we branch by splitting the rectangle with the largest upper bound along the coordinate i with maximum error $\epsilon_i = \hat{f}_i(x_i^\star(Q)) - f_i(x_i^\star(Q))$ into two subrectangles that meet at $\min(x_i^\star(Q), z_i)$. The concave approximation of f_i on the subrectangles is then exact at $x_i^\star(Q)$, so that each branch maximally reduces the error at the previous solution.

Convergence. In Appendix A, we show that the number of concave subproblems that must be solved to achieve accuracy ϵ^{des} is bounded by

$$\prod_{i=1}^n \left(\left\lceil \frac{(h_i(z_i) - h_i(l_i))\,(z_i - l_i)}{\epsilon^{\text{des}}/n} \right\rceil + 1 \right),$$

where $Q_{\text{init}} = (l_1, u_1) \times \cdots \times (l_n, u_n)$, $z_i = \arg\max_{[l_i, u_i]} h_i(x)$ for $i = 1, \ldots, n$, and $h_i : \mathbf{R} \to \mathbf{R}$ is any upper semi-continuous quasi-concave function such that $f_i(x) = \int_{l_i}^x h_i(t)\, dt$ (which always exists, if f_i is sigmoidal).

Pruning. If $\max_{Q \in \mathcal{Q}} L(Q) > U(Q')$, we know with certainty that the solution is not in rectangle Q', and we may safely delete rectangle Q' from the set $\mathcal{Q}$. In this case we say that rectangle Q' has been *pruned*. Conversely, we use the notion that a rectangle is *active* to mean that the possibility of finding the optimum in that rectangle has not been ruled out. The list of active rectangles maintained by the branch and bound algorithm then has an interpretation as the set of rectangles in which one is guaranteed to find at least one point for which the value of the objective differs from the optimal value by no more than $\max_{Q \in \mathcal{Q}} U(Q) - \max_{Q \in \mathcal{Q}} L(Q)$.

Note that our branching rule makes it unnecessary to explicitly prune the set $\mathcal{Q}$: if $U(Q') > \max_{Q \in \mathcal{Q}} L(Q)$, then the algorithm will terminate before rectangle Q' is selected.

6.1 Extensions

Sums of sigmoidal functions. It is worthwhile noting that the algorithm given above may be applied to other problem classes. The only requirement of the algorithm is the ability to construct a concave upper bound on the function to be maximized on any rectangle, which becomes provably tight as the size of the rectangle decreases. Hence this algorithm may be applied not only to sums of sigmoidal functions, but to sums of other functions with other convexity properties.

The concave envelope of a function f is, by definition, the smallest concave function majorizing f. So we can never arrive at a tighter approximation to f by computing the concave envelopes of some decomposition of f. That is, for any decomposition $f(x) = \sum_i g_i(x)$, we have $\hat{f}(x) \leq \sum_i \hat{g}_i(x)$. Hence the upper and lower bounds achieved will be tightest when the branch and bound algorithm is applied directly to f, rather than to some decomposition of f.

This observation has a number of simple consequences. We showed in the introduction that any univariate function with known regions of convexity and concavity can be written as a sum of sigmoidal functions. However, applying this algorithm directly to the original function might result in a more tight concave approximation, and hence faster convergence. Conversely, a difference of convex (DC) programming approach would be to decompose a sigmoidal function into one convex and one concave function, and apply the algorithm directly to these. However, the concave envelope of the convex function over the region will never be tighter than the concave envelope of the sigmoidal function. Hence the algorithm is unlikely to converge more quickly.

Penalty functions. Our method also extends to problems of the form

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} f_i(x_i) + \phi(x) \\ \text{subject to} & x \in \mathcal{C}, \end{array}$$

where ϕ is a concave reward function.

7 Numerical examples

In this section, we solve a number of sigmoidal programs using `SigmoidalProgramming`, an implementation written in Julia of the algorithm described in §6. This open source implementation is available online, with documentation, at

`https://github.com/madeleineudell/SigmoidalProgramming.jl`.

This implementation models the subproblem created at each node in the branch and bound tree using the JuMP optimization modeling library (Lubin and Dunning 2013), making it easy to substitute different solvers. In the examples below, all the convex subproblems are simply linear programs. By default, and in the examples below, `SigmoidalProgramming` uses GLPK (Makhorin 2006) to solve these LPs.

All experiments presented below were conducted on a desktop computer with a 3 GHz Intel Core 2 Duo processor, running Mac OS X 10.8.

7.1 Bidding example

As our first numerical example we consider an instance of the bidding problem given in §3.2.

In Figure 7, results are shown for an example in which

$$f_i(b_i) = \text{logistic}(\alpha_i b_i + \beta_i) - \text{logistic}(\beta_i),$$

with v_i drawn uniformly at random from the interval $[0, 4]$, $\alpha_i = 10$, and $\beta_i = -3v_i$, for $i = 1, \ldots, n$. We set the budget $B = .2\sum_{i=1}^{n} v_i$. Each graph represents one variable and its associated objective function (solid line) and concave approximation (dashed line) on the rectangle containing the solution. The solution $x_i^\star$ for each variable is given by the x coordinate of the red X. The rectangle containing the solution is bounded by the solid grey lines and the endpoints of the interval. For $n = 36$, the solution even after the first iteration is quite close to optimality, and the sigmoidal programming algorithm reaches a solution within $\epsilon = .01$ of optimality after solving only 17 convex subproblems. Notice that the solution lies along the concave part of the curve, or at the lower bound, for nearly every coordinate. This phenomenon illustrates a generic feature of allocation problems: the convex portion of the curve offers the best "bang-per-buck" (*i.e.*, utility as a function of the resources used), and so the optimal solution generally exploits this region fully or gives up on the coordinate entirely.

Table 1 gives the performance of the algorithm on bidding problems ranging in size n in producing a solution within $\epsilon = .01n$ of the global solution, with the other problem parameters drawn according to the same model as above. The table shows the number of subproblems solved and time to achieve a solution. As n increases, the time to solve each subproblem increases (since the linear program that we solve at each step is substantially larger), but the number of subproblems that must be solved to achieve the same relative error decreases, as we expect from the approximation guarantees given in §5.

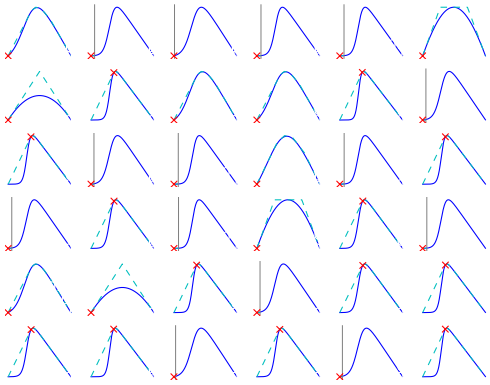

Figure 7: Solution for bidding example.

Table 1: Performance on bidding problems.

n	**subproblems**	**time (s)**
10	12	.10
20	28	.10
50	46	.39
100	1	.01
200	1	.03
500	1	.09
1000	1	.30
2000	1	.95
5000	1	5.09
10000	1	19.73

7.2 Network utility maximization

Our second example demonstrates the speed of the method on a typical allocation type problem. We consider the network utility maximization problem previously introduced in §3.4,

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} f_i(x_i) \\ \text{subject to} & Ax \leq c \\ & x \geq 0, \end{array}$$

where x represent flows, c are edge capacities, A is the edge incidence matrix mapping flows onto edges, and $f_i(x_i)$ is the utility derived from flow i.

Figure 8 shows the convergence of the algorithm for a network with $n = 500$ flows over 500 edges, with each flow using on average 2.5 edges each, where we use admittance functions as the utility functions f_i, and each edge has capacity 2.5. Within 14 iterations, the solution

is within 3% of optimality.

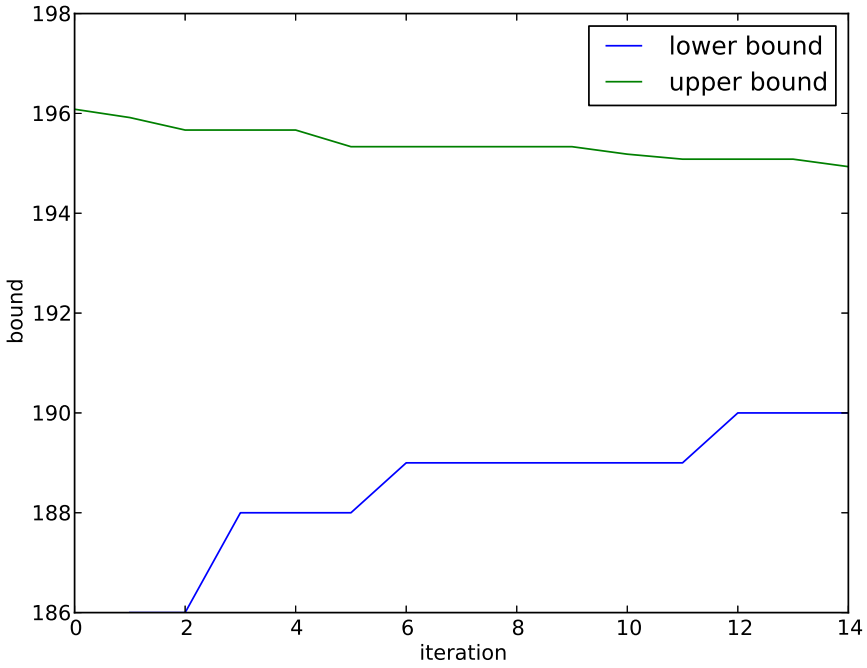

Figure 8: Convergence for NUM with admission function utilities, with $n = 500$ flows over 500 edges, with each flow using on average 2.5 edges each.

7.3 Political marketing example

We now consider an application of the targeted marketing problem (§3.1) to political marketing, in which the decision variables correspond to the positions that a politician takes on each particular issue, and the functions correspond to the expected number of votes for that politician from a given constituency. The goal of the politician is to choose positions on each issue to maximize his vote share.

The idea that politicians might opportunistically choose their positions in order to maximize their vote share is very old, going back to Downs (1957) and Hotelling (1929), in which voters' preferences are assumed to be distributed in some one-dimensional parameter space. Today, the *spatial theory of voting* expresses the idea that voters pick candidates based on the distance in "policy space" between the voter and candidate (Merrill and Grofman 1999). However, this model has generally proven intractable, since the nonconvexity of the politician's objective leads to a difficult (as we have shown) optimization problem, even with a relatively low-dimensional issue space (Roemer 2006). See, *e.g.*, Adams et al (2005) or Roemer (2006, 2004) for more detailed models of party competition based on the spatial theory

of voting.

Data. Problem data is generated using responses from the 2008 American National Election Survey (ANES 2008). Each respondent r in the survey rates each candidate c in the 2008 U.S. presidential election as having positions $y^{rc} \in [1,7]^m$ on m issues. Respondents also say how happy they would be $h^{rc} \in [1,7]$ if the candidate c won. We suppose a respondent would vote for a candidate c if $h^{rc} > h^{rc'}$ for any other candidate c'. If so, $v^{rc} = 1$ and otherwise $v^{rc} = 0$.

For each candidate c and state i, we predict that a respondent $r \in S_i$ in state i will vote for candidate c with probability $\text{logistic}((w_i^c)^T y^{rc})$, depending on the candidate's perceived positions y^{rc}. The parameter vector w_i^c is found by fitting a logistic regression model to the ANES data for each candidate and state pair.

Note that the data from the ANES 2008 survey is not meant to be representative of the population of the US on a state by state basis. It includes information on respondents from only 34 states, some with only 14 respondents.

Optimizing electoral votes. Suppose each state i has p_i votes, which are allocated entirely to the winner of the popular vote. Let $y \in \mathbf{R}^m$ denote the positions the politician takes on each of the m issues. The politician's *pandering* to state i is given by $x_i = w_i^T y$. Using our model, the expected number of votes from state i is

$$f_i(x_i) = p_i \Phi \left(\frac{\text{logistic}(x_i) - .5}{\sqrt{\text{logistic}(x_i)(1 - \text{logistic}(x_i))/N_i}} \right),$$

which is sigmoidal in x_i, where Φ is the normal CDF and N_i is the number of voters in state i. Hence the politician will win the most votes if y is chosen by solving

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n f_i(x_i) \\ \text{subject to} & x_i = w_i^T y, \quad i = 1, \ldots, n \\ & 1 \leq y \leq 7. \end{array}$$

Using SP, we find the optimal positions $y^\star$ for Obama (Table 2) to take in the 2008 election, with optimal pandering levels shown in Figure 9. We compare these with average position y_0 that respondents in the survey reported he took. Each graph represents one variable and its associated objective function (solid line) and concave approximation on the rectangle containing the solution (dashed line). The solution for each variable is given by the x coordinates of the red x. The boundary of the rectangle Q containing the solution is indicated by the vertical solid grey lines whenever the boundary is not either l_i or u_i. The positions are represented using the seven-point Likert scale described in Table 3.

8 Conclusion and future work

This paper introduced the sigmoidal programming problem, and discussed a number of settings in which these problems naturally occur. We showed that the problems are NP-

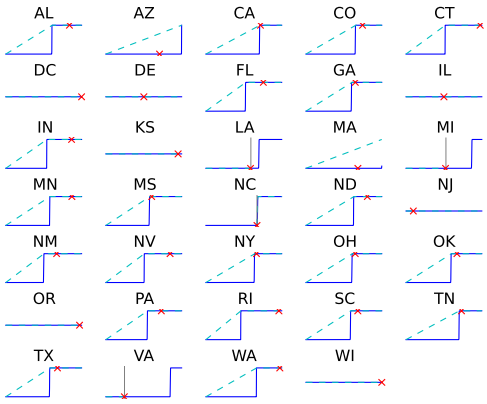

Figure 9: Optimal pandering for Obama in 2008.

Table 2: Optimal positions for Obama.

Issue	$y^\star$	y_0
spending/services	1.26	5.30
defense spending	1.27	3.69
liberal/conservative	1.00	3.29
govt assistance to blacks	1.00	3.12

Table 3: Encoding of positions.

Issue	**1**	**7**
spending/services	fewer services	more services
defense spending	decrease spending	increase spending
liberal/conservative	liberal	conservative
govt assistance to blacks	govt should help blacks	blacks should help themselves

hard in general, and even NP-hard to approximate, but that they may be easier in certain settings; in particular, when the number of constraints is small. We presented a simple framework for global optimization of sigmoidal programming problems using a branch and bound framework, taking advantage of the simple structure of sigmoidal functions to compute upper and lower bounds on the objective value via convex optimization, and showed that the method works well in practice, often requiring the solution of a single convex optimization problem to achieve satisfactory bounds.

A number of interesting questions remain for future work.

Finite convergence. Branch-and-bound algorithms for separable concave minimization problems can often be modified to converge to *exact* (rather than ϵ-approximate) solutions in a finite number of iterations; see, for example, Al-Khayyal and Sherali (2000) or Shectman and Sahinidis (1998). However, these algorithms (and proofs) rely on the property that solutions to the problems they study occur only at extreme points of the feasible set. In contrast, solutions to sigmoidal programs may occur in the interior. Is it possible to modify the algorithm presented here to ensure finite convergence?

Tighter convergence bounds. Our convergence proof does not make use of the known upper bound on the approximation error presented in §5. In fact, Udell and Boyd (2014) also observe that a solution x to the convex subproblem can be chosen such that $\epsilon_i(x)$ is nonzero for only m coordinates i. This fact can be used along with the same arguments presented in our convergence analysis to reduce the upper bound on the maximum number of subproblems to

$$2\prod_{i=1}^{n}\left(\left\lceil\frac{(h_i(z_i)-h_i(l_i))\,(z_i-l_i)}{\epsilon^{\mathrm{des}}/m}\right\rceil+1\right),$$

for $m < n$. Can further improvements be made using these approximation error bounds?

Better pruning and splitting rules. Our algorithm always splits the rectangle with the highest upper bound. Is it possible to achieve better convergence in practice using other rules to select the next rectangle to split?

A Convergence proof

In this section, we bound the number of convex subproblems that must be computed before obtaining a solution of some specified tolerance to the original sigmoidal programming problem (3).

Before beginning the proof, we remind the reader of a few of our definitions. For each sigmoidal function $f_i : [l_i, u_i] \to \mathbf{R}$, we define a bounded upper semi-continuous quasi-concave function $h_i : [l_i, u_i] \to \mathbf{R}$ such that

$$f_i(x) = f_i(l_i) + \int_{l_i}^{x} h_i(t)dt, \quad i = 1, \ldots, n.$$

(Such a function h_i always exists: continuity of f_i implies that h_i can be chosen to be upper semi-continuous; Lipshitz continuity of f_i implies that h_i is bounded; and that f_i is convex and then concave implies that h_i is quasi-concave.)

We let $z_i \in [l_i, u_i]$ be a point maximizing h_i over the interval $[l_i, u_i]$. (This is equivalent to saying that f_i is convex for $x < z_i$, and concave for $x > z_i$.) We define $\hat{f}_i : [l_i, u_i] \to \mathbf{R}$, as before, as the concave envelope of the function f_i, so that $\hat{f}_i(x) \geq f_i(x)$ for every $x \in [l_i, u_i]$. This envelope defines a unique point

$$w_i = \min\{x \in [z_i, u_i] \mid \hat{f}_i(x) = f_i(x)\}.$$

Furthermore, since f_i is sigmoidal,

$$\hat{f}_i(x) = f_i(x), \quad x \in [w_i, u_i].$$

Note in particular that $z_i \leq w_i$.

Recall the operation of the sigmoidal programming algorithm. Let $x^\star = x^\star(Q)$ be the solution to the problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n \hat{f}_i(x_i) \\ \text{subject to} & x \in \mathcal{C} \cap Q. \end{array}$$

So long as the total error is still greater than the desired tolerance ϵ^{des},

$$\sum_{i=1}^n \hat{f}_i(x^\star) - f_i(x^\star) \geq \epsilon^{\text{des}},$$

the algorithm proceeds by splitting the rectangle Q in order to attain a better approximation on each of the subrectangles. The rectangle Q is split along the coordinate i with the greatest error $\epsilon_i = \hat{f}_i(x_i^\star) - f_i(x_i^\star)$ into two smaller boxes that meet at $\min(x_i^\star, z_i)$.

We wish to show that this algorithm will terminate after a finite number of splits.

Notice that there can be no error in coordinate i if $x_i^\star \geq w_i$ or if $x_i^\star = l_i$, since $\hat{f}_i$ is the concave envelope of f_i. Furthermore, since f_i and $\hat{f}_i$ are both Lipshitz continuous, the error in the approximation is bounded by a constant (depending on h_i) times the minimum distance of $x_i^\star$ to l_i. We formalize this logic below to show that the size of the rectangles explored cannot be too small, which bounds the number of rectangles we may have to explore.

Error bounds. We first bound h_i above and below on $[l_i, w_i]$. The maximum of h_i, by definition, occurs at z_i:

$$\max_{x \in [l_i, w_i]} h_i(x) = h_i(z_i).$$

As for the minimum, the function h_i is quasi-concave, so its minimum must occur at either l_i or w_i. In fact, we can show it occurs at l_i. Let $g_i = \frac{f_i(w_i) - f_i(l_i)}{w_i - l_i}$. Note that g_i is a superdifferential of $\hat{f}_i$ at w_i, and that $g_i \leq h_i(w_i)$, since h_i has been chosen to be upper semi-continuous. So for $x \in [l_i, z_i]$,

$$\hat{f}_i(x) = f_i(l_i) + g_i(x - l_i) \leq f_i(l_i) + h_i(w_i)(x - l_i).$$

On the other hand, f_i is convex on $[l_i, z_i]$, so for $x \in [l_i, z_i]$,

$$f_i(x) \geq f_i(l_i) + h_i(l_i)(x - l_i).$$

Putting these together, we see

$$f_i(l_i) + h_i(l_i)(x - l_i) \leq f_i(x) \leq \hat{f}_i(x) \leq f_i(l_i) + h_i(w_i)(x - l_i),$$

so

$$h_i(l_i) \leq h_i(w_i).$$

We use these to bound f_i below on $[l_i, w_i]$:

$$\begin{aligned} f_i(x) &= f_i(l_i) + \int_{l_i}^{x} h_i(t)dt \\ &\geq f_i(l_i) + \left(\min_{s\in(l_i,z_i)} h_i(s)\right)(x - l_i) \\ &\geq f_i(l_i) + h_i(l_i)(x - l_i). \end{aligned}$$

Similarly, we bound $\hat{f}_i$ above on $[l_i, w_i]$:

$$\begin{aligned} \hat{f}_i(x) &= f_i(l_i) + \frac{\int_{l_i}^{w_i} h_i(t)dt}{w_i - l_i}(x - l_i) \\ &\leq f_i(l_i) + \frac{\int_{l_i}^{w_i} \max_{s\in(l_i,z_i)} h_i(s)dt}{w_i - l_i}(x - l_i) \\ &\leq f_i(l_i) + h_i(z_i)(x - l_i). \end{aligned}$$

Hence the error $\epsilon_i(x) = \hat{f}_i(x) - f_i(x)$ in the approximation to the ith objective function f_i at any point $l_i \leq x \leq w_i$ obeys

$$\begin{aligned} \epsilon_i(x) = \hat{f}_i(x) - f_i(x) &= \frac{\int_{l_i}^{w_i} h_i(t)dt}{w_i - l_i}(x - l_i) - \int_{l_i}^{x} h_i(t)dt \\ &\leq (h_i(z_i) - h_i(l_i))\,(x - l_i). \end{aligned} \tag{8}$$

We can make the same arguments integrating down from w_i instead of up from l_i to find

$$\epsilon_i(x) \leq (h_i(z_i) - h_i(l_i))\,(w_i - x) \leq (h_i(z_i) - h_i(l_i))\,(u_i - x). \tag{9}$$

Bounds on rectangle size. A rectangle can be split no further when the error on that rectangle is less than ϵ^{des}. A sufficient condition is $\epsilon_i \leq \epsilon^{\text{des}}/n$ for $i = 1, \ldots, n$.

Suppose that $x_i^\star < z_i$. Using equation (8), we see that we will only split on coordinate i if

$$\epsilon^{\text{des}}/n \leq \epsilon_i(x_i^\star) \leq (h_i(z_i) - h_i(l_i))\,(x_i^\star - l_i),$$

and using equation (9), we see that we will only split on coordinate i if

$$\epsilon^{\text{des}}/n \leq \epsilon_i(x_i^\star) \leq (h_i(z_i) - h_i(l_i))\,(u_i - x_i^\star),$$

Hence both rectangles produced by the split must satisfy

$$u_i - l_i \geq \frac{\epsilon^{\text{des}}/n}{h_i(z_i) - h_i(l_i)}.$$

This condition motivates the definition of the minimal side length,

$$\delta_i = \frac{\epsilon^{\text{des}}/n}{h_i(z_i) - h_i(l_i)}.$$

The method will never produce a rectangle with a side length smaller than this by splitting at a previous solution, *i.e.*, when $x_i^\star < z_i$.

Conversely, when $x_i^\star > z_i$, the method splits at z_i and may produce a left rectangle that is smaller. But this can only happen at most once for each coordinate.

Formally, we can show

Lemma 1. *Every side i of each rectangle produced by the method either contains z_i or has side length at least δ_i.*

Proof. The proof proceeds by induction. It is true for the original rectangle with sides $[l_i, u_i]$. If a rectangle not containing z_i is split along coordinate i, then $x_i^\star < z_i$, so the argument above shows that both rectangles produced by the split must satisfy $u_i - l_i \geq \delta_i$. If a rectangle containing z_i is split along coordinate i, then either $x_i^\star \geq z_i$, in which case both rectangles resulting from the split will contain z_i, or $x_i^\star < z_i$, in which case the argument above still applies to bound $u_i - l_i \geq \delta_i$. □

Hence the splits between l_i and z_i along dimension i must be spaced by at least δ_i from each other and from the lower endpoint l_i.

Bounds on number of convex subproblems. The maximum number of rectangles that can fit between l_i and z_i along any dimension i is thus

$$\left\lceil \frac{z_i - l_i}{\delta_i} \right\rceil,$$

tiling the interval (l_i, z_i) with rectangles of the minimum allowable side length. And the interval $[z_i, u_i]$ can never be split. So the maximal number of rectangles between l_i and u_i is

$$\left\lceil \frac{z_i - l_i}{\delta_i} \right\rceil + 1.$$

This implies the number of leaves l in the tree of rectangles can be at most

$$l \leq \prod_{i=1}^{n} \left(\left\lceil \frac{z_i - l_i}{\delta_i} \right\rceil + 1 \right).$$

A binary tree with l leaves has in total $2l - 1$ nodes.Hence the number of convex subproblems solved before the algorithm terminates is bounded by

$$\begin{aligned} 2l - 1 &\leq 2 \prod_{i=1}^{n} \left(\left\lceil \frac{z_i - l_i}{\delta_i} \right\rceil + 1 \right) \\ &= 2 \prod_{i=1}^{n} \left(\left\lceil \frac{(h_i(z_i) - h_i(l_i))\,(z_i - l_i)}{\epsilon^{\text{des}}/n} \right\rceil + 1 \right). \end{aligned}$$

In particular, the algorithm converges in a finite number of steps.

B Hardness of approximation

In this section, we show there can be no polynomial time algorithm to find an arbitrarily good approximate solution to any sigmoidal programming (SP) problem unless P=NP. Thus the algorithm we present for SP, which requires exponential time in some cases to achieve a sufficiently good approximate solution, cannot be substantially improved (unless P=NP). The proof uses a reduction of maximum independent set (MIS), a combinatorial optimization problem which is known to be NP-hard to approximate on a graph with n vertices within a factor $\rho > n^{1-\epsilon}$ for any $\epsilon > 0$ (Berman and Schnitger 1992, Hastad 1996, Beigel 1999), to SP. For completeness, we recall that x is an ρ-approximate solution to an maximization problem $\mathcal{P}$ with objective function $F \geq 0$ if x is feasible and

$$F(x) \geq \rho^{-1} F(x^\star),$$

where $x^\star$ is the solution to $\mathcal{P}$.

The hardness of approximation result relies on a guarantee (proved below) that any non-integral solution to SP can be rounded to an integral solution with strictly greater objective function, allowing an approximate solution to SP to be used to find an approximate solution to MIS. For a more thorough discussion of approximation algorithms and techniques for proving hardness of approximation results, we refer the reader to Hochbaum (1996), Vazirani (2001).

Recall the combinatorial optimization problem MIS: given a graph $G = (V, E)$, find a maximal subset of vertices $S \subseteq V$ such that no two vertices in S share an edge. We can write the problem as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n x_i \\ \text{subject to} & x_i + x_j \leq 1 \quad (i,j) \in E \\ & x \in \{0,1\}^n. \end{array} \tag{10}$$

We now describe the reduction of MIS to SP. The objective function in the SP formulation will include both the MIS objective and a term penalizing deviations from integrality. Let $f(x) = |x - 1/2| - 1/2$. Consider the problem

$$\begin{array}{ll} \text{maximize} & \gamma \sum_{i=1}^n f(x_i) + \sum_{i=1}^n x_i \\ \text{subject to} & x_i + x_j \leq 1 \quad (i,j) \in E \\ & x \in \{0,1\}^n. \end{array} \tag{11}$$

The first term in the objective vanishes if the solution x is integral, while the second term is just the MIS objective. Note that $\gamma f(x) + x$ is a sigmoidal function for any $\gamma \in \mathbf{R}$, so (11) is a sigmoidal programming problem. Define

$$F(x) = \gamma \sum_{i=1}^n f(x_i) + \sum_{i=1}^n x_i.$$

Theorem 2. *If $\gamma > 1$, then every local solution of (11) is integral, and any feasible point $x \in [0,1]^n$ for (11) can be rounded to a integral feasible point $x^{round} \in \{0,1\}^n$ such that $F(x) < F(x^{round})$. Hence, $x^\star$, the solution to (11), is integral.*

Proof. Consider a feasible point $x \in [0,1]^n$ for the problem (11) that is not integral, and the vector $x^{\text{round}} \in \{0,1\}^n$ that rounds each nonintegral entry x_i to the nearest integer. We choose the convention that $1/2$ is rounded down to 0, so that x^{round} is also a feasible point.

We wish first to show that $F(x) < F(x^{\text{round}})$. Let $\delta_i = |x_i - x_i^{\text{round}}|$, for $i = 1, \ldots, k$. By rounding x_i, we increase $f(x_i)$ by an amount δ_i, while decreasing x_i by at most δ_i. This is true for every $i = 1, \ldots, k$, so

$$F(x^{\text{round}}) - F(x) \geq (\gamma - 1) \sum_{i=1}^{n} \delta_i.$$

That is, for $\gamma > 1$, x^{round} always achieves a larger objective value than x (unless $x^{\text{round}} = x$). This shows that any local solution to the problem (11) will always be integral, and so in particular the global solution will be integral. □

Theorem 2 allows us to show that the optimal value of (11) is equal to the optimal value of (10). Using the theorem, we know $x^\star \in \{0,1\}^n$ and $f(x_i^\star) = 0$, for $i = 1, \ldots, n$, so the problem (11) has the same solution and optimal value as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n x_i \\ \text{subject to} & x_i + x_j \leq 1 \quad (i,j) \in E \\ & x \in \{0,1\}^n, \end{array}$$

which is exactly the MIS problem.

This argument shows that sigmoidal programming is NP-hard. We now show it is also NP-hard to approximate, by using the property, shown above, that the objective value for a solution always increases when the solution is rounded to integrality.

Theorem 3. *It is NP-hard to compute an ρ-approximate solution to the sigmoidal programming problem*

$$\begin{array}{ll} \textit{maximize} & \sum_{i=1}^n f_i(x_i) \\ \textit{subject to} & x \in \mathcal{C}, \end{array}$$

where $f_i(x)$ is sigmoidal $i = 1, \ldots, n$, for $\rho > n^{1-\epsilon}$ with any $\epsilon > 0$.

Proof. We prove this result by showing that an ρ-approximate solution to the sigmoidal programming problem gives a polynomial time computable ρ-approximate solution to MIS.

Let $x^\star$ be the solution to (11), and consider an ρ-approximate solution x^{apx} to (11) with $F(x^{\text{apx}}) \geq \rho^{-1} F(x^\star)$ for some $\rho \in (0,1)$. Let x^{round} denote the solution obtained by rounding the entries of x^{apx}. Then x^{round} still satisfies the constraints of (11), and $F(x^{\text{round}}) \geq F(x^{\text{apx}})$ by theorem (2) if we choose $\gamma > 1$. Thus x^{round} is an ρ-approximate solution to (11), since $F(x^{\text{round}}) \geq F(x^{\text{apx}}) \geq \rho^{-1} F(x^\star)$. But notice that x^{round} also satisfies the constraints of (10). That is, x^{round} gives an independent set of size $F(x^{\text{round}})$. Then since the optimal values and solutions to (10) and (11) are equal,

$$\sum_{i=1}^{n} x_i^{\text{round}} = F(x^{\text{round}}) \geq F(x^{\text{apx}}) \geq \rho^{-1} F(x^\star) = \rho^{-1} \sum_{i=1}^{n} x_i^\star,$$

so x^{round} is an ρ-approximate solution to (10).

That is, we may use our ρ-approximate solution x^{apx} to (11) to compute in polynomial time an ρ-approximate solution x^{round} to (10), which is known to be NP-hard for $\rho > n^{1-\epsilon}$ with any $\epsilon > 0$ (Berman and Schnitger 1992, Hastad 1996, Beigel 1999). □

This theorem shows that there can be no polynomial time approximation scheme (PTAS) for sigmoidal programming, because it is NP-hard to approximate SP to within a factor better than $n^{1-\epsilon}$.

C Acknowledgements

The authors are grateful to Jon Lee, Ernest Ryu, and to a number of anonymous reviewers for their helpful comments and suggestions. This work was supported by National Science Foundation Graduate Research Fellowship program (under Grant No. DGE-1147470), the Gabilan Stanford Graduate Fellowship, the Gerald J. Lieberman Fellowship, and the DARPA X-DATA program.

References

Achterberg T (2009) SCIP: solving constraint integer programs. Mathematical Programming Computation 1(1):1–41

Adams J, Merrill III S, Grofman B (2005) A unified theory of party competition: a cross-national analysis integrating spatial and behavioral factors. Cambridge University Press

Al-Khayyal F, Sherali H (2000) On finitely terminating branch-and-bound algorithms for some global optimization problems. SIAM Journal on Optimization 10(4):1049–1057

Andersen M, Dahl J, Vandenberghe L (2013) CVXOPT: A Python package for convex optimization, version 1.1.5. URL `abel.ee.ucla.edu/cvxopt`

ANES (2008) The ANES 2008 time series study dataset. `www.electionstudies.org`

Balakrishnan V, Boyd S, Balemi S (1991) Branch and bound algorithm for computing the minimum stability degree of parameter-dependent linear systems. International Journal of Robust and Nonlinear Control 1(4):295–317, DOI 10.1002/rnc.4590010404

Balas E (1968) A note on the branch-and-bound principle. Operations Research 16(2):442–445

Beigel R (1999) Finding maximum independent sets in sparse and general graphs. In: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, pp 856–857

Belotti P (2009) CoUEnnE: a users manual. Technical Report

Berman P, Schnitger G (1992) On the complexity of approximating the independent set problem. Information and Computation 96(1):77–94

Boyd S, Vandenberghe L (2004) Convex Optimization. Cambridge University Press

Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS) 2:303–314, 10.1007/BF02551274

D'Ambrosio C, Lee J, Wächter A (2012) An algorithmic framework for MINLP with separable non-convexity. In: Mixed Integer Nonlinear Programming, Springer, pp 315–347

Downs A (1957) An economic theory of political action in a democracy. Journal of Political Economy 65(2):135–150

Falk J, Soland R (1969) An algorithm for separable nonconvex programming problems. Management Science 15(9):550–569

Fazel M, Chiang M (2005) Network utility maximization with nonconcave utilities using sum-of-squares method. In: 44th IEEE Conference on Decision and Control, pp 1867 – 1874, DOI 10.1109/CDC.2005.1582432

Friedman M, Savage L (1948) The utility analysis of choices involving risk. The Journal of Political Economy 56(4):279–304

Hastad J (1996) Clique is hard to approximate within $n^{1-\epsilon}$. In: Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, IEEE, pp 627–636

Henrion D, Lasserre JB (2005) Detecting global optimality and extracting solutions in gloptipoly. In: Henrion D, Garulli A (eds) Positive Polynomials in Control, Lecture Notes in Control and Information Science, vol 312, Springer Berlin Heidelberg, pp 293–310, DOI 10.1007/10997703\X

Hochbaum D (1996) Approximation algorithms for NP-hard problems. PWS Publishing Co.

Horst R, Pardalos P, Van Thoai N (2000) Introduction to global optimization. Kluwer Academic Publishers

Hotelling H (1929) Stability in competition. The Economic Journal 39(153):pp. 41–57

Kahneman D, Tversky A (1979) Prospect theory: An analysis of decision under risk. Econometrica: Journal of the Econometric Society pp 263–291

Karp R (1972) Reducibility among combinatorial problems. Springer

Lasserre J (2001) Global optimization with polynomials and the problem of moments. SIAM Journal on Optimization 11:796–817

Lawler E, Wood D (1966) Branch-and-bound methods: A survey. Operations Research 14(4):699–719

Lipp T, Boyd S (2014) Variations and extensions of the convex-concave procedure

Lubin M, Dunning I (2013) Computing in operations research using Julia. arXiv preprint arXiv:13121431

Makhorin A (2006) GLPK (GNU linear programming kit)

McCormick G (1976) Computability of global solutions to factorable nonconvex programs: Part i: Convex underestimating problems. Mathematical programming 10(1):147–175

Merrill S, Grofman B (1999) A Unified Theory of Voting: Directional and Proximity Spatial Models. Cambridge University Press

Misener R, Floudas C (2014) ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations. Journal of Global Optimization 59(2-3):503–526

Parrilo P (2003) Semidefinite programming relaxations for semialgebraic problems. Mathematical programming 96(2):293–320

Roemer J (2004) Modeling party competition in general elections. Cowles Foundation Discussion Paper

Roemer J (2006) Political competition: Theory and applications. Harvard University Press

Sahinidis N (2014) BARON 14.3.1: Global Optimization of Mixed-Integer Nonlinear Programs, *User's Manual*

Shectman J, Sahinidis N (1998) A finite algorithm for global minimization of separable concave programs. Journal of Global Optimization 12(1):1–36

Sturm J (1999) Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optimization methods and software 11(1-4):625–653

Tawarmalani M, Sahinidis N (2002) Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications, vol 65. Springer Science & Business Media

Tawarmalani M, Sahinidis N (2005) A polyhedral branch-and-cut approach to global optimization. Mathematical Programming 103:225–249

Toh KC, Todd M, Tütüncü R (1999) SDPT3 — a MATLAB software package for semidefinite programming, version 1.3. Optimization Methods and Software 11(1-4):545–581

Tversky A, Kahneman D (1992) Advances in prospect theory: Cumulative representation of uncertainty. Journal of Risk and uncertainty 5(4):297–323

Udell M, Boyd S (2014) Bounding duality gap for separable functions. arXiv preprint arXiv:14104158 `1410.4158`

Vazirani V (2001) Approximation algorithms. Springer-Verlag New York, Inc., New York, NY, USA