Work Summary for Fall 2017

Jonathan Robert Perlstein
jrperl{at}cs.stanford.edu

December 10, 2017

1 Overview

This document serves as the principal summary of my research work completed during the Fall 2017 quarter. My work during this quarter and my ongoing research both focus on one primary objective: developing and implementing techniques to solve the Blind Decoding problem using a vertex-hopping methodology inspired by the simplex algorithm. For the Fall 2017 quarter I undertook two tasks in this vein. First, I studied the Blind Decoding problem itself in much greater depth, to include preparing and delivering a presentation concerning the problem and my Summer 2017 research to the Stanford Wireless Systems Laboratory. Second, I worked with Tom Dean to begin development and implementation of the algorithms needed to port the Blind Decoding solver to using an efficient vertex-hopping methodology.

In this section I provide an overview of my two tasks for the Fall quarter, in addition to connecting this work to my previous and future research. This document is intended to be read in conjunction with the slide deck from the WSL Group presentation and the in-development prototype code implementation of the simplex-style solver for Blind Decoding, both of which are provided separately.

Summer / Fall Intercessional Period During the Summer / Fall intercessional period, I rewrote the internals of my general-purpose simplex solver to optimize speed. This completed my familiarization study of simplex from the perspective of learning the algorithm by writing my own implementation.

Principal Problem Researched During Fall Quarter After completing the generic simplex solver, we began work on applying simplex techniques to the Blind Decoding problem. When necessary, standard simplex overcomes the issue of finding an initial basic feasible solution (BFS) by generating and solving a related linear program, which provides a starting vertex for the original LP. We tried adapting this technique to Blind Decoding; however, the existence of nonsingular vertices at which the gradient is undefined significantly complicates the problem. Using the conventional simplex methodology would require that we encode the nonsingularity restriction directly as linear constraints, and we do not see an obvious means of creating such an encoding directly from the problem parameters. After further work, we devised a new technique for finding an initial BFS. In certain instances this technique also involves solving a smaller LP that is directly related to the original problem; however, our methodology ensures that the returned vertex corresponds to a nonsingular matrix. I describe the problem in further detail, including our progress to date, in Section 2. The current prototype code is provided in `initial_bfs.py`.

Further Study of Blind Decoding Problem and Presentation to WSL Group To gain the problem-specific knowledge needed to assist in upgrading the Blind Decoding solver internals, I performed a more detailed study of the *Blind Decoding* paper and related theoretical concepts. At the end of Week 6 I presented the current state of the Blind Decoding research to the Stanford Wireless Systems Laboratory. This presentation included a summary of previous work conducted by Thomas Dean, Mary Wootters, and Andrea Goldsmith; my research on the $n = 5$ case from the Summer of 2017; and a brief look at our current and future work on this problem. I provide a short summary of this presentation in Section 3. I also include the full slide deck separately in `presentation.pdf`.

Future Research I provide an overview of my planned work for the Fall / Winter intercessional period in Section 2. Further work will be planned during Week 1 of the Winter quarter.

2 Beginning the Port of the Blind Decoding Solver to Simplex

In this section I will describe some general issues related to porting the Blind Decoding solver from the current algorithm, which is an interior-point based method, to a vertex-hopping technique inspired by simplex.

2.1 Motivation for Simplex

As described in the *Blind Decoding* paper, the current solution utilizes an interior-point based methodology. While the algorithms presented in *Blind Decoding* do work, they have inherent limitations based on the structure of the underlying problem. Recall the mathematical definition of the Blind Decoding problem.

$$\underset{\mathbf{U}}{\text{maximize}} \quad \log|\det \mathbf{U}| \tag{1}$$

$$\text{subject to} \quad ||\mathbf{U}\mathbf{y}_j||_\infty \leq M + c \cdot \sigma,\ j = 1, \ldots, k \tag{2}$$

from which we can determine:

- There are only $2kn$ constraints, all of which are *linear*.
- The objective function has a closed-form gradient computable in $O(n^3)$ time: $(U^{-1})^T$ [proven in *Blind Decoding*].
- All solutions to the Blind Decoding problem lie at vertices [also proven in *Blind Decoding*].

The inherent mismatch between the current algorithms and the problem geometry now is apparent. To render the interior-point methods feasible, the algorithms in *Blind Decoding* use a barrier function to recast the constrained optimization problem as an unconstrained optimization problem. This methodology does succeed in forcing gradient descent to stay within the feasible region; however, we are using a *barrier function* to compute a problem where all solutions provably lie at *vertices* of the feasible region. A barrier function by definition pushes away from the edges of the feasible region, while a simplex-style vertex pivoting solution seems to fit this problem more closely.

2.2 Initial Approach with Simplex

2.2.1 Brief Simplex Overview

The simplex algorithm operates by hopping among vertices of the feasible region. Solving a linear program via simplex necessitates converting the LP to standard form and then creating a simplex tableau.[1] Once the tableau is created, the simplex pivoting functionality [hopping from one vertex to another] operates as follows.

- At each vertex of the feasible region, simplex divides the variables into the categories of `basic` and `nonbasic`. Note that the full set of variables includes both the original variables in the linear constraints plus a separate slack or surplus variable for each constraint. Nonbasic variables must have the value 0, while basic variables can take any nonnegative value. During a pivot step, the algorithm selects a currently nonbasic variable that is to become basic. The value of this variable is increased[2] until a different constraint becomes tight, which will occur when a different variable [one that was basic prior to the pivot] reaches 0. The variable that moves from nonbasic to basic is termed the `entering` variable, while the variable that moves from basic to nonbasic is termed the `leaving` variable.
- The pivoting process is similar in spirit and implementation to the main iterative step in Gauss-Jordan elimination. A simplex pivot effectively moves the solution from one vertex to a different vertex of the feasible region.
- In a conventional implementation of simplex aimed at solving LP's, any pivot that increases the value of the objective function[3] can be chosen. My reference simplex implementation [built in Summer 2017] chooses pivots at random from among those that increase the value of the objective.

[1]There are multiple different ideas of simplex standard form. My implementation more closely resembles that shown in CLRS than that shown in Papadimitrou's Combinatorial Optimization, though both approaches are fundamentally equivalent.

[2]In standard form, all variables must be nonnegative, so unrestricted real-valued variables are replaced by the difference of two nonnegative variables before the pivoting process begins.

[3]Throughout this document I will assume that the LP of concern is a maximization LP, which is true of the Blind Decoding problem.

- Simplex thus climbs the gradient via vertex hopping, stopping once it reaches a vertex that is a local optimum. Since the convexity of linear programs guarantees that every local optimum is a global optimum, simplex terminates at an optimal solution assuming that the LP has a feasible region that is both nonempty and bounded.[4]

2.2.2 Preliminary Considerations in Using Simplex for Blind Decoding

We identified three areas that we must address in converting the Blind Decoding problem to a form in which the simplex style vertex pivoting methodology can be leveraged to find a solution.

Variables / Constraints Converting the constraints of the Blind Decoding algorithm shown in Equation (2) to simplex involves straightforward techniques. The n^2 entries of U are unrestricted real-valued variables. In our LP formulation, we thus use the difference of two nonnegative variables for each variable of $U : u_i = x_{2i} - x_{2i+1}$.

The constraints on the L_∞ norm of each Uy_j mean that:

$$\forall i \in \{1, \dots, n\} \,.\, \forall j \in \{1, \dots, k\} : |\langle u_i, y_j \rangle| \;\le\; M = 1$$

Using the standard linear programming technique of simulating the constraint $|a| \le b$ with the pair of constraints $a \le b$ and $-a \le b$, we obtain the $2kn$ applicable linear constraints. We therefore conclude that the variables and constraints of the Blind Decoding problem can be addressed using conventional linear programming tools.

Objective Function While the constraints of the Blind Decoding problem do not require any deviations from standard linear programming techniques, the objective function is not convex, let alone linear. The *Blind Decoding* paper demonstrates why this function behaves like a convex function, and our previous work provides theoretical assurance for this behavior extending to the $n = 5$ case. In terms of choosing an entering variable at each simplex pivot, however, the method used by conventional simplex cannot be applied since it depends on the linearity of the objective function. We briefly address this issue in Section 2.3 but need to perform further work in this area.

Initial Basic Feasible Solution As previously mentioned, simplex requires that LP's be converted to a standard form where every variable is restricted to being nonnegative. Conventional simplex normally begins at the origin; however, there is no guarantee in any arbitrary LP that the origin is feasible, and simplex operates by hopping among vertices of the feasible region. In instances where the origin falls outside the feasible region, conventional simplex finds an initial basic feasible solution through solving a different, closely related linear program.[5] In the context of the Blind Decoding problem, however, this methodology is insufficient. For the gradient to be defined, the initial BFS vertex must correspond to a *nonsingular* matrix. Our efforts to address this issue are described throughout the remainder of this section.

2.3 Focus on Finding an Initial BFS

The requirement that the initial basic feasible solution correspond to a nonsingular matrix is a unique feature of the Blind Decoding algorithm that is not inherently captured by its constraints when encoded as an LP. As such, we need to devise a specific algorithm for finding an initial BFS, which is explained below.

2.3.1 Algorithm Description

Note: *The algorithm described here in Section 2.3.1 and the failure case solution described in 2.3.2 are credited to Tom Dean, and the entirety of Section 2.3.1 borrows directly from his work. I assisted with code simulations related to Section 2.3.2. My principal contributions to the initial BFS research are the simplex-related techniques described in the subsections that follow Section 2.3.2.*

As an overview, the algorithm attempts to find a nonsingular BFS by solving n^2 consecutive optimization problems, each in 1 dimension. Starting from a random point in the feasible region, the algorithm moves along the line of maximum gradient until it reaches the feasible region boundary. The algorithm then re-computes the gradient at the new point, projects the gradient onto the nullspace of the active constraints, and moves along this projection again

[4]While simplex in theory is in `EXPTIME` and is not in P, it has been demonstrated that simplex runs in polynomial time [and with comparatively good constant factors] on nearly all non-pathological inputs.

[5]This related program is not the dual program. Rather, it is a specially designed LP that either provides a feasible vertex of the original LP or provably demonstrates the original LP to have no feasible region.

to the problem boundary. Since moving in the nullspace of active constraints does not disturb any already-tight constraints, the algorithm will reach a vertex in the n^2 iterations. A more detailed specification of the algorithm follows:

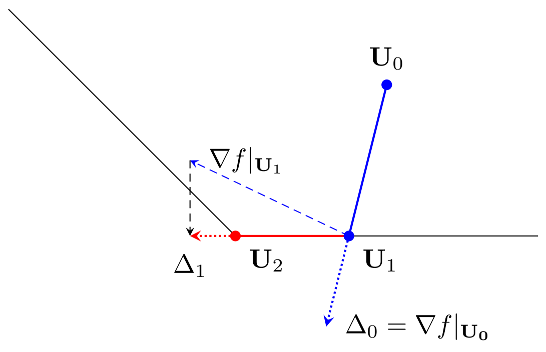
- Pick an initial value of U within the feasible region.
- Repeat n^2 times:
  - Compute $\Delta = \nabla(U)$. Project Δ onto the nullspace of the active constraints obtaining Δ_p.
  - Solve the optimization problem:

$$\begin{array}{ll} \underset{t \in R}{\texttt{maximize}} & f(U + t\Delta_p) \\ \texttt{subject to} & t \geq 0, ||(U + t\Delta_p)Y||_\infty \leq 1 \end{array}$$

The optimization problem of finding the value of t that exactly reaches the problem boundary is efficiently solved by finding the smallest absolute value element of $(|UY| - 1)/\Delta_p$ [where division is performed elementwise]. In terms of time complexity, the high-order asymptotic term stems from computing the gradient at each step, which requires $O(n^3)$ time for $O(n^2)$ steps yielding an $O(n^5)$ algorithm.

Figure 1 provides a visual depiction of the first two steps of the algorithm. Note specifically that the path taken from U_1 to U_2 overlays the projection of the gradient onto the nullspace of active constraints [at U_1 there is exactly one active constraint].

Figure 1: BFS Finding Algorithm


$\mathbf{U}_0$
$\nabla f|_{\mathbf{U}_1}$
Δ_1
$\mathbf{U}_2$
$\mathbf{U}_1$
$\Delta_0 = \nabla f|_{\mathbf{U_0}}$

2.3.2 Failure Case and Remediation

Empirical evidence derived through computer simulation demonstrates that the algorithm described in Section 2.3.1 successfully finds a suitable BFS in the $n = 4$ / $k = 5$ case with probability ≈ 0.5. Though we could simply rerun the algorithm and succeed after only 2 iterations in expectation, we chose to pursue a deterministic solution. The underlying theory indicates that at higher dimensions a simple randomized algorithm will have intolerably low per-iteration success probability, so we chose to invest time in derandomization in the more tractable $n = 4$ case with the intention of applying our technique to higher dimensions.

The failure case usually takes the following form:

- There are 4 columns of Y [denote the matrix formed by just these columns as Y'] such that $det(UY') = 8$. The entries of the other column of UY are in $\{-1, 0, +1\}$ instead of $\{-1, +1\}$.
- The gradient is zero throughout the entire nullspace of the active constraints, so the projection of the gradient onto the nullspace is nonexistent.

We are attempting to remediate this problem through the following methodology:

- Temporarily drop the offending column of Y, so we are left with an $n = 4$ / $k = 4$ problem.
- Turn this smaller problem into a pseudo-LP; since we already are at a vertex in this smaller LP, use simplex-style techniques to hop to an optimal vertex.

- At each vertex, verify whether the temporarily dropped column of Y would be violated by the proposed pivot. If so, reject the pivot. If not, take the pivot. Remembering rejected pivots requires some amount of bookkeeping not associated with standard simplex. Backtracking requires that state be maintained as restorable snapshots.

2.3.3 Incorporation of Simplex Techniques to Find Initial Nonsingular BFS

As described in the previous section, our potential methodology for overcoming the failure case itself relies on simplex techniques. I therefore focused on developing these techniques in the context of finding the initial BFS, with intention of applying the same prototype code toward the larger simplex implementation on the full Blind Decoding problem once the initial BFS is found.

Tableau generation Creating the simplex tableau involves translating the Blind Decoding variables and constraints into standard linear programming form, as earlier described in Section 2.2.2. In the $n = 4$ / $k = 4$ case for the subprogram we are using to find the initial BFS, we have $2kn = 2 \cdot 4 \cdot 4 = 32$ linear constraints. Since $U \in \mathbb{R}^{n\times n} = \mathbb{R}^{4\times 4}$, there are 16 variables of U, each of which is unrestricted and thus must be replaced by the difference of two nonnegative variables, yielding 32 total variables. Since our problem has a maximization objective, standard from requires that all constraints be of $\leq$ form, so we also need 32 slack variables, giving 64 total variables.

While generating the constraints itself is straightforward, the failure case of the algorithm described in Sections 2.3.1 and 2.3.2 will simply provide a $U \in \mathbb{R}^{n\times n}$ and a reduced $Y' \in \mathbb{R}^{n\times n}$. In order to utilize the pivoting technique of simplex, each constraint row of the tableau must have *exactly* one basic variable with a nonzero coefficient [a row can have any number of nonbasic variables, since these must be assigned value 0]. Therefore, we must perform steps similar in nature to Gauss-Jordan elimination to transform our initial tableau into a valid state. Our methodology is as follows:

- Recall that we have $||UY'||_\infty \leq 1$ where $U, Y' \in \mathbb{R}^{n\times n}$. Consider each row of U separately. For an individual row u_i, we have that $\forall j \in \{1, \dots, n\} : \langle u_i, y_j\rangle \leq 1 \wedge -\langle u_i, y_j\rangle \leq 1$. This gives $2n$ inequalities for each row.
- Still in the context of a fixed u_i, consider a fixed y_j as well. There are two relevant inequalities as mentioned above: $\langle u_i, y_j\rangle \leq 1$ and $-\langle u_i, y_j\rangle \leq 1$. Note that at any vertex of the feasible region *exactly one* of these inequalities is tight.
- For the inequality which is *not* tight, the corresponding slack variable must have value $1 - (-1) = 2 \neq 0$, so the slack variable is the sole basic variable in that row of the simplex tableau.
- For the inequality in the pair which is tight, the slack variable will have value 0 at this vertex. This allows us to choose exactly one pair of non-slack x variables [that correspond to a single U variable][6] to have a nonzero coefficient in this specific row only. We first normalize the coefficients of the pair of x variables to have values $+1, -1$ in this row, and we then perform a Gauss-Jordan style elimination step to zero out the coefficients of these two x variables in every other of the $2n$ rows where these variables appear in the original tableau.
- Recall that there are $2n$ constraints relevant to any row of the original U matrix. Among these constraints, there are $2n$ total different x variables, corresponding to the n variables in a single row of U. There are exactly n rows where the slack variable has value 0 [the n tight constraints], and each such row can be used to zero out the coefficients of one pair of x-variables in all other rows. Within each pair of x variables, at most one will have nonzero value at any point in the feasible region, since we can express all real values of u_i in the equation $u_i = x_{2i} - x_{2i+1}$ by setting at most one of the x variables to a positive value.
- Our methodology thus produces exactly one basic variable with a nonzero coefficient for each of the $2kn$ constraints. For each of the kn pairs of constraints, we must eliminate one variable pair from $2n$ equations, each of which contain $2n$ nonzero entries. While there may be some additional efficiencies that I currently do not see, I believe that the lower bound of $\Omega(kn^3)$ [since $k \geq n$ so this is $\Omega(n^4)$] may be tight.

Pivoting At any vertex of the feasible region, the following will hold:

$$\forall i \in \{1, \dots, n\} \, . \, \forall j \in \{1, \dots, k\} : |\langle u_i, y_j\rangle| = 1$$

[6]Recall that each variable u_i is replaced by the difference of two nonnegative variables $x_{2i} - x_{2i+1}$.

Since $|\langle u_i, y_j \rangle| \leq 1$ converts to the two linear constraints $\langle u_i, y_j \rangle \leq 1$ and $-\langle u_i, y_j \rangle \leq 1$, at any vertex of the feasible region exactly one of these two constraints will be tight, and the other will have a slack of 2. Pivoting to an adjacent vertex of the feasible region involves flipping which of the two constraints is tight for exactly one (i, j) pair. This corresponds to flipping the sign of exactly one entry in the matrix $UY \in \{-1, +1\}^{n \times k}$.

Once we have created the tableau we will have kn rows of the form

$$x_{2q} + x_{2q+1} = 2$$

where x_{2q} and x_{2q+1} are the slack variables corresponding the the two equations for the constraint $|\langle u_i, y_j \rangle| \leq 1$. When we execute a pivot, we do not change this equation at all [we only switch which of x_{2q} and x_{2q+1} has value 2 and which has value 0], but we do change every equation that contains the *entering* variable [the one whose value is changing from 0 to 2].

To execute a pivot, we simply perform a Gauss-Jordan style step on each equation in the specific group of $2n$ equations corresponding to the row of UY in which the sign flip will occur. Within this group of $2n$ equations, exactly n of the equations will have a nonzero coefficient for the *entering* variable. We add a multiple of the row shown above, which will effectively replace the entering variable with the leaving variable in the targeted row, and will change the right hand side accordingly. This maintains the required simplex invariant that every row of the tableau have exactly one basic variable with a nonzero coefficient.

Checking the dropped constraint At each pivot in our algorithm on the reduced LP, we check to ensure that the extra Y column that we originally dropped is not violated by this pivot. If the extra Y column is violated, we consider this pivot to be invalid and backtrack. We do not attempt a peek-pivoting solution, where we perform a partial pivot and attempt to determine at an earlier stage if the pivot will violate the dropped constraint. We do not see an obvious means of implementing peek pivoting in a manner more efficient than executing and checking the full pivot.

Backtracking Already in this initial problem of finding a nonsingular BFS, our algorithm requires the ability to backtrack. This capability represents a substantive addition to the vertex-hopping methodology of simplex. When simplex operates on a linear program, backtracking is never necessary; however, in our algorithm, backtracking will be necessary in multiple contexts. Implementing backtracking requires as least the following:

- We must save the necessary state, which involves saving a snapshot of the simplex tableau itself. Computationally this requires very little time and space for any reasonable backtracking depth.
- We must maintain a set of visited vertices, similar to any graph traversal algorithm.

We intend to implement backtracking using a depth-first-search type of technique.

Leveraging the gradient In standard simplex, each possible entering variable - and the effect of choosing it on the objective function - is plainly visible through cursory examination of the objective function row of the simplex tableau. This clarity is not present in Blind Decoding. Pivoting requires choosing an entering variable from among the slack variables that is currently 0 and changing it to 2 [while changing its mirror variable to 0]. The gradient, however, is in terms of U, where each of the n^2 entries of U corresponds to exactly 2 nonnegative variables in the LP [i.e. $u_i = x_{2i} - x_{2i+1}$]. Thus in the tableau the gradient is *not* in terms of the potential entering variables. We are considering how to leverage the gradient to select entering variables efficiently.

2.4 Code Prototype

We have implemented a prototype for finding the initial basic feasible solution, which is included as `initial_bfs.py`. Our implementation contains functionality for all of the areas described throughout Section 2.3 with the exception of leveraging the gradient to make informed pivoting decisions. Additionally, our backtracking functionality is very primitive at this point. Previous mathematical work by Tom Dean demonstrates that the maximum necessary backtracking depth for both finding an initial BFS and for finding a global optimum in the $n \leq 5$ case is 1. Therefore, our current simplistic implementation of backtracking will suffice for this early stage but will need to be expanded substantially in later phases of the project.

2.4.1 Near-Term Next Steps

Our near-term implementation priorities are as follows:

- Determining how to select pivots efficiently is the most important next step from where we are. [Goal for Fall / Winter intercessional period].
- Once we combine the pivoting method with the basic backtracking capability, the problem of finding an initial nonsingular BFS will be solved. [Goal for Fall / Winter intercessional period].
- Once we have the initial nonsingular BFS, we can use vertex-hopping to solve the Blind Decoding problem. All of the adaptations to simplex described in Section 2.3 apply to the larger problem as well. Our code prototype already has factored and abstracted the functionality for solving the optimization problem that underlies Blind Decoding. Therefore, we readily can set a new optimization problem that uses the initial nonsingular BFS, where this second problem is the full instance of Blind Decoding, with all columns of Y [all received symbols] retained. Our current prototype will need to be generalized, since for simplicity the tableau code assumes that $k = n$, which is not the case once we consider the full Blind Decoding problem.

Our longer-term implementation goals are as follows:

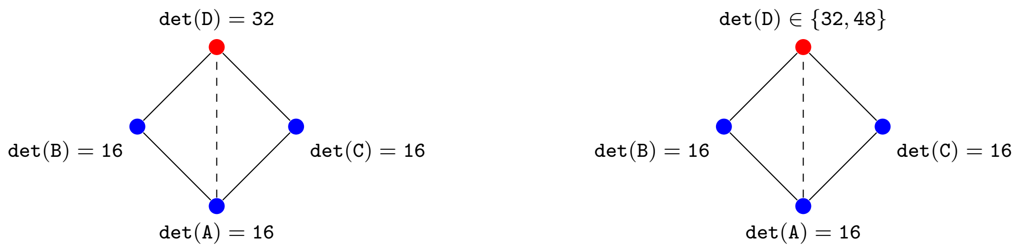
- Once the prototype code is fully functional, we will port it to a fast compiled language and optimize the code for performance. We intend to take this step for the $n = 4$ and $n = 5$ cases before exploring higher dimensional cases.
- Previous mathematical proofs by Tom Dean demonstrate that the necessary backtracking depth remains theoretically tractable for larger values of n. Since we know that there are local optima for $n \geq 6$, we will need to implement careful backtracking algorithms. We intend to explore this space once the $n \leq 5$ cases are fully functional with optimized code.

3 Presentation to Wireless Systems Laboratory

I delivered a presentation concerning my research in support of the Blind Decoding problem on Friday, 2017/11/03, to the Stanford Wireless Systems Laboratory. This presentation covered the following points.

- Background and motivation for the Blind Decoding approach, to include the initial mathematical vision and the extremely promising empirical results that spurred interest in this technique.
- Explanation of my research concerning the $n = 5$ case.
  - The computational difficulties presented by the problem geometry and the methods by which I devised an efficient solution.
  - The unexpected results that the matrices $\{-1, +1\}^{5x5}$, viewed as a graph with edges between each pair of matrices at Hamming distance 1, still form a single connected component even after all matrices of determinant 0 are removed. This was not the case for $n < 5$.
  - The further unexpected result that the left hand side of Figure 2 does *not* hold in the $n = 5$ case, along with my determination that the right hand side of Figure 2 *does* hold [I also described the programming techniques that I utilized to efficiently determine this fact through computer simulation].

Figure 2: Gradient in $n = 5$ Case



- Future research direction using the simplex algorithm to hop between vertices, since previous work demonstrated that all solutions to the Blind Decoding problem lie at vertices, suggesting that a customized simplex implementation could be extremely efficient.

The full slides of this presentation are provided in a separate document, `presentation.pdf`.