Conic Sampling: An Efficient Method for Solving Linear and Quadratic Programming by Randomly Linking Constraints within the Interior

Oliver Serang[1,2]*

1 Department of Neurobiology, Harvard Medical School, Boston, Massachusetts, United States of America, 2 Department of Pathology, Boston Children's Hospital, Boston, Massachusetts, United States of America

Abstract

Linear programming (LP) problems are commonly used in analysis and resource allocation, frequently surfacing as approximations to more difficult problems. Existing approaches to LP have been dominated by a small group of methods, and randomized algorithms have not enjoyed popularity in practice. This paper introduces a novel randomized method of solving LP problems by moving along the facets and within the interior of the polytope along rays randomly sampled from the polyhedral cones defined by the bounding constraints. This *conic sampling* method is then applied to randomly sampled LPs, and its runtime performance is shown to compare favorably to the simplex and primal affine-scaling algorithms, especially on polytopes with certain characteristics. The conic sampling method is then adapted and applied to solve a certain quadratic program, which compute a projection onto a polytope; the proposed method is shown to outperform the proprietary software Mathematica on large, sparse QP problems constructed from mass spectometry-based proteomics.

Citation: Serang O (2012) Conic Sampling: An Efficient Method for Solving Linear and Quadratic Programming by Randomly Linking Constraints within the Interior. PLoS ONE 7(8): e43706. doi:10.1371/journal.pone.0043706

Editor: Jérémie Bourdon, Université de Nantes, France

Received June 12, 2012; **Accepted** July 25, 2012; **Published** August 27, 2012

Copyright: © 2012 Oliver Serang. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Funding: The author has no support or funding to report.

Competing Interests: The author has declared that no competing interests exist.

* E-mail: Oliver.Serang@Childrens.Harvard.edu

Introduction

Linear programming involves optimizing a linear objective function subject to a collection of linear constraints. LP problems are frequently encountered throughout many disciplines, both on their own and as approximations to more complex problems. Linear programming has recently been applied to image reconstruction [1,2], modeling Markov decision processes [3], and graphical models [4,5].

Formally, LP requires optimizing an n dimensional linear function $f^T x$ over a feasible region defined by k affine inequality constraints $A_i x \leq b_i$. Each row of the $k \times n$ matrix A, along with the corresponding element in the column vector b, defines a single halfspace, and the feasible region, denoted Ω, is composed of the intersection of these halfspaces. Thus, any LP problem can be stated as follows:

$$\text{Minimize } g(x) = f^T x$$

$$\text{Subject to } Ax \leq b$$

The solution to the LP problem consists of a point x^* with minimal $g(x^*)$.

Finding a feasible point $x_{init} \in \Omega$ can itself be written as a linear program that maximizes feasibility (this is called a "two phase" approach). Alternately, feasible points can be found during optimization by creating a trivially feasible problem with augmented slack variables δ, and simultaneously minimizing $f^T x + M 1^T \delta$. If M is a large enough constant, the penalty $M 1^T \delta$ will be driven to 0 at an optimum (known as the "Big M" method) [6].

Simplex Methods

The first practical algorithm for solving LP problems, the simplex algorithm [7], was described in 1947. This algorithm embeds the feasible region into a simplex, and then takes steps along vertices on the simplex that decrease the objective function. These steps correspond to movement along the edges of the feasible region, by which one bounding constraint is exchanged for another. When several possible adjacent vertices allow a decrease in the objective value (as is frequently the case), then a pivot rule is used to resolve which will be taken. The simplex algorithm has been shown to have worst-case exponential behavior on certain problems [8] but is efficient in practice, and is still a popular method for solving linear programs. Randomized simplex algorithms, which employ stochastic pivot rules, have been shown to evade exponential behavior [9], but in practice tend to perform worse than deterministic variants. Pseudocode for the steepest-edge and randomized simplex methods implemented for comparison are provided in Algorithm 0, with subroutines as Algorithms 0–0. The simplex variant described and used in this manuscript requires the point 0 to be in the feasible region; however more sophisticated simplex methods, (*e.g.* the parametric self-dual simplex method [6]) operate using the same basic motivation, but can be used to solve LPs that are not trivially feasible (by implicitly transforming the LP using a method similarly motivated to the Big M method described above, thus manipulating the objective value and the feasibility). These simplex variants can also be used with stochastic pivot rules, and can alternate between primal and dual steps.

Algorithm 1 The simplex algorithm. The input is a collection of linear constraints, represented by A and b designed so that $A0 \preceq b$ (if not possible, the Big M method mentioned above, is used), the linear objective function (f), and $PivotColumn$, a function to compute the pivot column from a collection of edges. The algorithm halts when it reaches x^*, a point at which the objective function attains its minimum value in the feasible region. If no such point exists, then the algorithm reports that the problem is unbounded and halts. In the pseudocode, all variables are passed by reference.

```
procedure SIMPLEX(A, b, f, PivotColumn)
    T ← [ f^T  0  0 ; A  I  b ]                    ▷ Initialize tableau
    basisRows ← [2,...,k+1]
    basisCols ← [n+1,...,n+k]
    loop
        edges ← T[0]
        if edges ⪰ 0 then
            break: Optimum reached
        end if
        pc ← PivotColumn(edges)
        col ← column(T, pc)
        if col ⪯ 0 then
            HALT: problem unbounded
        end if
        lastCol ← column(T, n+k+1)        ▷ Get final column (one-indexed arrays)
        pos ← {i : col[i] > 0.0}
        pr ← pos[argmin([lastCol[i] / col[i], i ∈ pos])]        ▷ Minimum ratio test
        Pivot(T, pr, pc, basisRows, basisCols)
    end loop
    x* ← [0, i = 1,...,n]
    for j = 1 to len(basisCols) do
        if basisColumns[j] ≤ n then                ▷ not a slack variable
            x*[basisCols[j]] = T[basisRows[j], n+k+1]
        end if
    end for
    return (x*)
end procedure
```

Algorithm 2 The simplex pivot subroutine. The inputs are a tableau T, row index row, column index col, array of basis row indices $basisRows$, and an array of basis column indices $basisCols$. The routine performs a pivot on the matrix T and updates $basisRows$ and $basisCols$. In-place modification operations are shown using C++ notation (*e.g.* $x += y$ to indicate an in-place $x \leftarrow x + y$ operation).

```
procedure Pivot(T, row, col, basisRows, basisCols)
    T[row] /= T[row,col]
    for i = 1 to k do
        if i ≠ row then
            T[i] -= T[i,col] * T[row]
        end if
    end for
    ind ← i : basisRows[i] = row
    removeIndex(basisRows, ind)
    removeIndex(basisCols, ind)
    append(basisRows, row)
    append(basisRows, col)
end procedure
```

Generalizations of Simplex Methods

Other geometric methods share similarities to simplex methods and move along the convex hull of the polytope; however, these methods are not restricted to moving along vertices, and so they can be viewed as generalizations of simplex approaches. One such approach is the geometrically motivated gravity descent method [10], which simulates the descent of a very small (radius ϵ) sphere of "mercury" to the minimum of the polytope. As the sphere descends, the walls of constraints it encounters create a reciprocal force, essentially projecting the objective vector to glide along the facets of the polytope. At each

Algorithm 3 Simplex pivot rule subroutines. The input is a collection of edges. Each routine returns the index of the column chosen for pivoting. One of these routines is provided as the final argument to the simplex method.

```
procedure SteepestPivotColumn(edges)
    return (argmin edges)
end procedure
procedure RandomPivotColumn(edges)
    return (randomValue(edges))
end procedure
```

iteration, finding the new steepest direction requires solving a small quadratic program (QP) on the set of bounding "active" constraints. Aside from a few subtleties (*e.g.* progressively decreasing the radius of the sphere if it becomes stuck in the vee of two very close facets), the method proceeds in its QP-based descent until the objective value cannot be decreased (as shown by the QP solution).

Interior Point Methods

In contrast with simplex methods, which traverse adjacent vertices of the polytope, interior point methods remain in the strict interior and asymptotically approach a solution in an iterative manner. Interior point methods terminate once the current solution reaches a predefined precision, and then may optionally use other methods to descend to the nearest vertex and reach an exact solution.

The ellipsoid method was the first algorithm proven to solve the LP to a predetermined precision in a polynomial number of steps [11]. The algorithm successively finds the ellipsoid of minimal volume that contains the intersection of the feasible region and the halfspace requiring the objective value to not increase. In each iteration, a step is taken to the center of the containing ellipsoid, ensuring an exponential decay in the volume of ellipsoids in the series. Although the algorithm converges close to an optimal solution in polynomial time, in practice it is not competitive with the simplex algorithm.

The advent of Karmarkar's algorithm for LP marked a shift in focus from simplex-based algorithms to interior point methods [12,13], as well as their primal-dual adaptations [14]. Karmarkar's algorithm is guaranteed to solve LP problems in polynomial time, asymptotically converging to a desired precision; however, unlike the ellipsoid algorithm, variants of Karmarkar's interior point method can be fast in practice. The method applies a logarithmic barrier function in lieu of constraints, and takes steps to simultaneously maximize the feasibility and the optimality. A simplified version, the primal affine-scaling method [15,16], has worst-case exponential behavior but is practically efficient, especially for large, highly constrained LPs. The primal affine-scaling method repeatedly takes steps in the direction of steepest feasible improvement to the objective. It does so in each iteration by inscribing an ellipsoid into the constaints limiting the local feasible region, optimizes the objective function over the hull of the ellipsoid, and finally then takes a step in the direction of that optimum. The ellipsoid is constructed in a manner that scales the space to afford an equal slack to all nearby constraints; this scaling prevents a single nearby constraint from strongly influencing the direction chosen [6]. Pseudocode for the affine-scaling method implemented for comparison is provided in Algorithm 4. Note that sparse vector implementations have low overhead for products between matrices and diagonal matrices, with runtime similar to matrix-vector products.

Algorithm 4 The affine-scaling algorithm. The input is a collection of linear constraints, represented by A and b, the linear objective function (f), a feasible point x_{init} such that $Ax_{init} \preceq b$, and a small value $\epsilon = 10^{-07}$ used for termination. The algorithm halts when it reaches x^*, a point at which the objective function attains its minimum value in the feasible region. If no such point exists, then the algorithm reports that the problem is unbounded and halts.

```
procedure AffineScaling(A, b, f, x_init)
    x ← x_init
    obj ← f^T x
    γ ← 0.9
    loop
        v ← b − Ax
        vNegSquared ← [1/v[i]^2, i = 1,...,k]
        hx ← (A^T diagonalMatrix(vNegSquared) A)^{-1} f
        hv ← −Ahx
        st ← {i : hv[i] < 0.0}
        if |st| = 0 then
            HALT: problem is unbounded
        end if
        α ← γ min([−v[i]/hv[i], i ∈ st])
        x ← x + αhx
        if f^T x / obj < ε then
            return (x)
        end if
        obj ← f^T x
    end loop
end procedure
```

Additional Randomized Methods

Following the advent and success of the randomized simplex pivot rules, other stochastic algorithms emerged for solving LP. Seidel's algorithm algorithm randomly downsamples from the set of all constraints [17]. The subproblems will either yield feasible, optimal solutions, or, if not, will indicate that at least one removed constraint bounds the optimum x^*. In this manner, the algorithm winnows down set of extreme (*i.e.* important, bounding) constraints. The Matousek/Sharir/Welzl algorithm uses a similar approach, but utilizes further information from the subspace spanned by the basis of currently known extreme constraints [18], thus establishing a new subexponential bound for LP. These algorithms show great promise for future use for have uncovered novel theoretical knowledge of polytopes and the LP problem, but have not yet enjoyed the broad success of simplex methods and interior point methods in practical application.

A Novel Method that Randomly Links Vertices within the Interior

The simplex algorithm and interior point methods are among the most commonly applied algorithms for LP, due to their simplicity of implementation and their efficiency. Randomized variants of these algorithms are generally thought to perform inefficiently relative to deterministic algorithms. Hence, randomized algorithms are usually only mentioned in the context of avoiding exponential average time performance on pathological problems and are unpopular in practice.

This manuscript describes a simple, geometrically motivated algorithm for LP that randomly samples from the interior of the feasible region in a manner that randomly selects from the set of superior vertices. This stochastic optimization algorithm, named *conic sampling*, is both simple and efficient. For LPs with certain characteristics, the conic sampling algorithm is demonstrated to roughly match or exceed the efficiency of the simplex and primal affine-scaling algorithms, particularly for highly constrained, sparse problems.

Methods

The proposed algorithm descends to a vertex by taking steps through the interior and projecting orthogonal to constraint rows A that are encountered. Once a vertex is reached, the algorithm randomly samples from the cone made by these accumulated constraints, finding a direction that improves the objective function and satisfies the accumulated constraints. The algorithm terminates if no direction exists that would satisfy the constraints and improve the objective function.

The two-dimensional example in Figure 1 illustrates the basic idea of the conic sampling method applied to an LP problem. Beginning at point A, the algorithm proceeds in a stepwise fashion through points B, C and D. Each step brings the algorithm closer to convergence at point E. In general, the algorithm follows the vector that minimizes the objective function while obeying all of the currently active constraints. Such a move will typically involve traversal across the facet of the polytope that encloses the feasible region. In this process, the procedure will sometimes encounter a vertex, such as C, in which direct movement in the direction decreasing the objective function is not possible.

When a vertex is reached, the set of halfspaces defined by the active constraints, intersected with the halfspace corresponding to non-decreasing objective function, yields a cone of possible legal moves. As the name implies, the conic sampling algorithm randomly selects a ray from within this cone and advances in the selected direction until a new constraint is encountered. Note that, in some degenerate cases, the sampled ray will yield a move of length zero. In this situation, a new ray is sampled until a non-zero move can be made. A pseudocode description of the procedure is given as Algorithm 5, with subroutines as Algorithms 6–8.

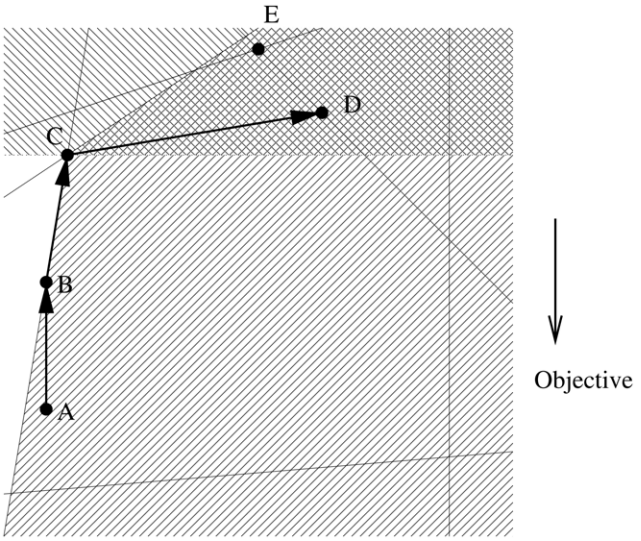

Figure 1. Illustration of conic sampling in two dimensions. The algorithm begins at point A, follows the objective function until it encounters a constraint at B, and then proceeds along the edge of the feasible region to C. At C, the algorithm randomly samples a ray from the cone produced by the intersection of halfspaces defined by improvement on the objective and the polyhedral cone defined by the active constraints. In the figure, this cone is indicated by overlapping shading. Following the sampled ray leads to D. The algorithm continues in a similar fashion, descending to fixation at the minimum point E in the following iteration.
doi:10.1371/journal.pone.0043706.g001

The motivation for the conic sampling algorithm is that, given the set of vertices of the polytope in a total ordering by objective value (vertices with equal objective value are never visited sequentially, and so vertices with identical objective can be ignored)

$$V_1, V_2, V_3, \ldots$$

$$f^T V_1 > f^T V_2 > f^T V_3 \ldots$$

at each iteration starting from vertex V_i, the algorithm will sample from the vertices V_j such that $j > i$. If this sampling is performed uniformly over the remaining candidate vertices then, on average, at each iteration half of the vertices will be eliminated. Although the sampling performed by the conic sampling algorithm is not necessarily uniform, it seems to be close to uniform in practice for certain polytopes. In general, one large advance that occurs at an early iteration early will winnow out many candidates for the next iteration, increasing the chance of choosing the optimum.

The random forward direction subroutine finds the spanning vectors of the cone made by a basis of bounding (also called "active") constraints and then generates a random conic combination of these vectors that lies in the positive halfspace of f. For the cone $K = \{q : \forall i \in M, A_i q \leq 0\}$, then it has spanning vectors v_1, $v_2 \ldots$, such that $K = \{\sum_i \alpha_i v_i, \alpha \leq 0\}$. These spanning vectors can be found by projecting every constraint orthogonal to every other constraint. If no vector in the cone can improve the objective, then optimality can be shown. This is similar to trying the n different pivots using the basis M in the simplex algorithm; however, instead of taking the best ray as chosen by some pivot rule, the candidate rays are combined so that the resulting direction is not necessarily restricted to an edge of the polytope.

Algorithm 5 The conic sampling algorithm. The input is a collection of linear constraints, represented by A and b, the linear objective function (f), and a feasible point x_{init} such that $Ax_{init} \preceq b$. The algorithm halts when it reaches x^*, a point at which the objective function attains its minimum value in the feasible region. If no such point exists, then the algorithm reports that the problem is unbounded and halts.

procedure CONICSAMPLING(A, b, f, x_{init})
  $x \leftarrow x_{init}$
  loop
    $M \leftarrow \{\}$
    AdvanceToFixation(A, b, f, x, M) ▷ Algorithm 7
    $H \leftarrow \{i : A_i x = b_i\}$
    $(p,t) \leftarrow$ RandomForwardDirection(f, $[A_i : i \in M]$, f) ▷ Algorithm 8
    if t **then**
      return (x) ▷ Optimum reached
    end if
    $x \leftarrow$ Advance(A, b, p, x) ▷ Algorithm 6
  end loop
end procedure

Algorithm 6 Conic sampling advance subroutine. The input is the column n-vector p. The subroutine takes the longest allowed step along p that will not violate any constraints. If such a step does not exist, then the problem is unbounded and the algorithm halts. The point x is updated to reflect the furthest advance along p, and the index of the first constraint hit is returned.

for $i = 1$ to k **do**
  if $A_i p \neq 0$ **then**
    set λ_i so that $A_i(\lambda_i p + x) = b_i$
  else
    $\lambda_i \leftarrow 0$
  end if
end for
if $\forall i, \lambda_i \geq 0$ **then**
  HALT: Problem is unbounded
end if
$\gamma \leftarrow \max_{i:\lambda_i<0} \lambda_i$
$j \leftarrow min(\{i : \lambda_i = \gamma\})$ ▷ When multiple constraints are hit at once, choose the first lexicographically.
$x \leftarrow x + \gamma p$ ▷ Move to the first constraint hit.
return j

Algorithm 7 Conic sampling advance to fixation subroutine. The point x is advanced along projections of f orthogonal to a set of constraints that have been hit during the Advance subroutine. This set grows until f has no component orthogonal to the set of constraints $A_i : i \in M$.

$p \leftarrow f$
while $\| p \| \neq 0$ **do**
  $i \leftarrow$ Advance(A, b, p, x)
  $M \leftarrow M + \{i\}$
  $Q \leftarrow$ Orthogonalize($[A_i : i \in M]$)
  $p \leftarrow f - \sum_j \frac{Q_j f}{\| Q_j \|^2} Q_j^T$ ▷ Project f orthogonal to the bounding A_i.
end while
if M is lexicographically maximal among bounded constraints **then**
  HALT: x is optimum
end if

Algorithm 8 Conic sampling random forward direction subroutine. The input is an $m \times n$ matrix C, with row rank m. It returns a column n-vector u s.t. $Cu \preceq 0$ and $f^T u > 0$, and a boolean t indicating whether such a vector other than $u = 0$ exists.

```
t ← false
d_+ ← 0
d_- ← 0
for i = 1 to m do
    Q ← Orthogonalize([C_j : j ≠ i])
    v_i ← C_i^T − Σ_j (Q_j C_i / ||Q_j||^2) Q_j^T        ▷ Project C_i orthogonal to the other C_j.
    if f^T v_i > 0 then
        d_+ ← d_+ + Uniform([0,1]) v_i
        t ← true
    else
        d_- ← d_- + Uniform([0,1]) v_i
    end if
end for
ψ_max = (−f^T d_-) / (f^T d_+ − f^T d_-)
ψ = Uniform([0, ψ_max])
u = (1 − ψ) d_+ + ψ d_-        ▷ Choose a convex combination of d_+ and d_- s.t. f^T u > 0.
return (u, t)
```

Proof of Convergence and Optimality

In the following, it is proven that the AdvanceToFixation subroutine terminates and that the ConicSampling algorithm terminates only when optimal. Furthermore, the ConicSampling algorithm is guaranteed to terminate, because the algorithm will advance to an improved vertex with each iteration.

Lemma 1. *The* AdvanceToFixation *subroutine terminates in no more than* n *iterations.*

Proof. Each iteration calls the Advance subroutine, which must either halt the algorithm or add an element to the set M. Because M begins empty, inductively assume that the set of vectors $A_i : i \in M$ is initially linearly independent. The direction p is orthogonalized to these vectors. If the Advance subroutine adds an element j to M, then $p^T A_j > 0$. Therefore, A_j is not a linear combination of the current set of vectors $\{A_i : i \in M\}$, to which p is orthogonal. Thus the set of vectors must be linearly independent, and can therefore contain no more than n vectors.

Lemma 2. *If the algorithm halts, then* x *attains an optimum objective value.*

Proof.

$$\Omega = \{u : Au \leq b\}$$

$$\Theta = \{u : \forall i \in H, A_i u \leq b_i\}$$

$$\Omega \subseteq \Theta$$

Because Θ is convex,

$$\forall y \in \Theta, g(y) \geq g(x) + \nabla g(x)^T (y - x)$$

The set H denotes constraints that support x:

$$\forall i \in H,\ A_i x = b_i$$

$$\forall y \in \Theta,\ \forall i \in H,\ A_i y \leq b_i$$

$$\forall y \in \Theta,\ \forall i \in H,\ A_i y - A_i x \leq 0$$

$$\forall y \in \Theta,\ \forall i \in H,\ A_i (y - x) \leq 0$$

Then for any value of $y \in \Theta$, $y - x$ is in a polar cone.

$$K = \{u : \forall i \in H,\ A_i u \leq 0\}$$

$$y \in \Theta \leftrightarrow y - x \in K$$

The algorithm terminates when.

$$\forall y - x \in K,\ \nabla g(x)^T (y - x) \geq 0$$

$$\forall y \in \Theta,\ \nabla g(x)^T (y - x) \geq 0$$

$$\forall y \in \Theta,\ g(y) \geq g(x) + \nabla g(x)^T (y - x)$$

$$\forall y \in \Theta,\ g(y) \geq g(x)$$

And finally, since Θ is a superset of Ω,

$$\forall y \in \Omega,\ g(y) \geq g(x)$$

On nondegenerate problems (problems where each vertex has the minimum number of bounding constraints), the method is guaranteed to advance to a new vertex point with each iteration; the number of vertices is, at most, exponential with n.

Degenerate problems are slightly trickier; there it is sufficient to guarantee that the set of constraints bounding a particular vertex are added to M in lexicographic order. By visiting them in this order, the method is guaranteed to add every linearly independent combination of bounding constraints to M; if the vertex is not the optimum, one of these combinations is guaranteed to have a non-empty polyhedral cone and advance. This is trivially observed via Bland's anticycling pivot rule [19], which is guaranteed to find an edge advancing from any suboptimal vertex (and thus result in at least one non-empty polyhedral cone for sampling). Thus, in the worst case, the algorithm will behave as a less efficient version of the simplex method using Bland's pivot rule.

Adaptation to Quadratic Programming

Because the conic sampling algorithm can occupy polytope vertices, facets, and the interior, it can be applied to quadratic programs (QPs), whereas the simplex methods, only occupying vertices, would not be not appropriate. In particular, the method has great potential for efficiently finding the projection of a vector x_0 onto a polytope or polyhedral cone:

$$\text{Minimize } \|x - x_0\|_2^2$$

$$\text{Subject to } Ax \leq b$$

Projections onto polytopes prove a useful exemplar subclass of quadratic programs, because they do not require conjugate gradient descent and can be trivially adapted from the existing algorithm. The modified advance subroutine, which has few changes, given in Algorithm 0, has two main changes: First, the initial local gradient (denoted f_x for simplicity, in lieu of all instances of f above) is initialized as $x_0 - x$ (and re-evaluated each time x changes). Second, the algorithm does not advance to the limiting constraint if the projection along the free axis p lies inside the polytope.

Algorithm 9 Modified conic sampling advance subroutine. This method adapts the LP variant to solve projections on polytopes. A column n-vector p is provided as input. The subroutine takes the optimal allowed step along p that will not violate any constraints. The point x is updated to reflect the optimal advance along p, and then the local gradient f_x is computed as $x - x_0$. If a limiting constraint is encountered, the index of the first constraint hit is returned. Otherwise, the algorithm halts at an optimum.

```
for i = 1 to k
    if A_i p ≠ 0 then
        set λ_i so that A_i(λ_i p + x) = b_i
    else
        λ_i ← 0
    end if
end for
γ* ← ((x_0 − x)^T p) / ||p||_2^2
if ∀i, λ_i ≥ 0
    x ← x + γ*
    HALT: x is optimum
end if
γ ← max_{i:λ_i<0} λ_i
γ ← min(γ, γ*)
j ← min({i : λ_i = γ})      ▷ When multiple constraints are hit at once, choose the first lexicographically.
x ← x + γp                  ▷ Move to the first constraint hit.
f_x ← x_0 − x               ▷ Update the local gradient
return j
```

Proof of Convergence and Optimality for Projection onto Polytopes

Using the above proof of optimality for the LP conic sampling algorithm, it is straightforward to show that the slightly modified algorithm converges to an optimum.

Lemma 3. *The modified* ConicSampling *algorithm terminates.*

Proof. As before, the algorithm always improves the objective value with each iteration (in the case of degenerate constraints, the anticycling rule is used once again). There are finitely many linearly independent sets of constraints to reach with each iteration, and so the algorithm must terminate.

Lemma 4. *The algorithm always finds an optimum in finitely many steps.*

Proof. The algorithm has two cases in which it halts (projections cannot be unbounded because the minimum distance is at most 0): The first case is identical to the LP version, where the algorithm halts when the intersection of halfspaces formed by the local gradient f_x and adjacent constraints is empty. The second case occurs when the projection of x_0 on the set of active constraints (in the modified advance subroutine) lies in the feasible region Ω; if this is the case, then the projection onto a superset more restrictive feasible region is itself feasible, indicating optimality by definition. Note that this third case of optimality would not be sufficient to demonstrate optimality for general QPs, (but slower conjugate gradient descent can be used).

Implementation

The conic sampling algorithm, in addition to the simplex method and primal affine-scaling method, are implemented by using a vector library written in C++ (code freely available upon request). Although these implementations are not necessarily expected to be scale competitively on large problems (due to both the efficiency and numeric stability of the implementations), their runtimes should have the same order runtime as more sophisticated variants. In these simulations, proportionality constants should not bias one algorithm over others, because they all designed and optimized with the same vector code base.

Throughout the code, numerical comparisons, particularly when comparing values to zero, were performed using a very small tolerance $\epsilon = 10^{-10}$. This increases the stability when values deviate from zero due to accumulated numerical error.

Sparse Vector Math

Methods are compared using a shared code base, and verified using two separate vector data structures appropriate for problems of different size. The first vector data structure, which is most appropriate for small problems (and, hence is used for the preliminary runtime analysis), is a dual sparse-dense data structure. The dual sparse-dense vector data structure indexes the nonzero indices, but also stores the entire vector (zeros included) in a contiguous block to prevent copying (*e.g.* for instance during an insertion of a value into the middle of a vector), and permit efficient random access. Although storing the entire vector as a contiguous block substantially increases the space requirement, it results in very fast in-place vector operations, particularly products between very sparse and very dense vectors. It is worth noting that, aside from loading the matrix A (necessarily performed by all algorithms), the simplex tableau is operated on in a sparse manner, and so entries with values of approximately zero do not influence the runtime).

The second vector library, which only stores the nonzero indices and their values, is less efficient on smaller problems (due to slower in-place operations and slower vector products as described above), but can be applied to much larger problems, where the dual sparse-dense vector data structure become far too memory intensive. This vector library is used to benchmark the algorithms on highly sparse LPs where k is far larger than n.

All algorithms substantially exploit sparsity in f, A, and b. The sparse simplex implementation stores the tableau in a data structure of sparse vectors for each row. More complex variants of the sparse simplex method implemented for these experiments (*e.g.* the Forrest-Tomlin method and Reid's modified Bartels-Golub method [20,21]) may have increased numeric stability, but their complexity introduces an increased risk of overhead that may also unfairly penalize the simplex method.

The Orthogonalize routine, likewise, operates using a matrix comprised of an array of sparse vectors. It takes B as input and returns a matrix Q such that rowspan $(Q) =$ rowspan(B) and the rows of Q are orthogonal. In practice, this computation can be accomplished efficiently using modified Gram-Schmidt; adding an individual row to B and recomputing Q can be performed in $\mathcal{O}(n^2)$ by exploiting the preexisting orthogonality and only adjusting the newly inserted row. Similar considerations have been made when orthogonalizing in the RandomForwardDirection subroutine, where overlapping sets of vectors are repeatedly orthogonalized. Sherman-Morrison updating and sparse LU updating could be used alternatively to compute the projections with the same result, and, possibly, with greater numeric precision. Regardless, more efficiency would be possible by preferrentially ordering the rows so that more sparse rows are on the top, thus preserving their sparsity.

Results

All runtimes were taken on the same computer using UNIX user time and were programmed using the same code base. Polytopes were chosen so that the feasible region includes 0 in order to not unfairly penalize the simplex method. C++ programs were compiled with gcc-4.7 using -O3 optimizations.

Preliminary Runtime Analysis on Random LPs

The runtime of the conic sampling algorithm was compared to the simplex and primal affine-scaling algorithms. The simplex method was implemented to employ different pivot rules: steepest edge and random edge (Algorithm 3). Algorithm runtimes are compared by using the same sparse vector code base and shared common functions (for these preliminary experiments, dual sparse-dense vectors were used, due to their highly efficient in-place access and product operations).

The algorithms were run on a set of randomly generated linear programs, which were made by uniformly (in [0,1000]) sampling the values of respective rows and values of of A and b such that $b \succ 0$, ensuring the feasible region trivially contained $x = 0$. Additional constraints require non-negativity of all elements of x (these constraints are not added to the simplex methods, because they would not influence the result and would increase the simplex runtimes). Each element of the objective vector was likewise chosen uniformly.

The runtimes for different sizes of problems with varying levels of sparsity (*i.e.* varying percents of elements in A, b, and f equal to zero) are shown in Figure 2. The nth dimension was *not* randomly set to zero so that no zero-norm constraints would be selected. LPs with a high ratio of k/n and a high level of sparsity are particularly efficient with conic sampling. Although by no means comprehensive, these random LPs demonstrate the existence of polytopes for which the conic sampling method reliably outperforms simplex variants and affine-scaling.

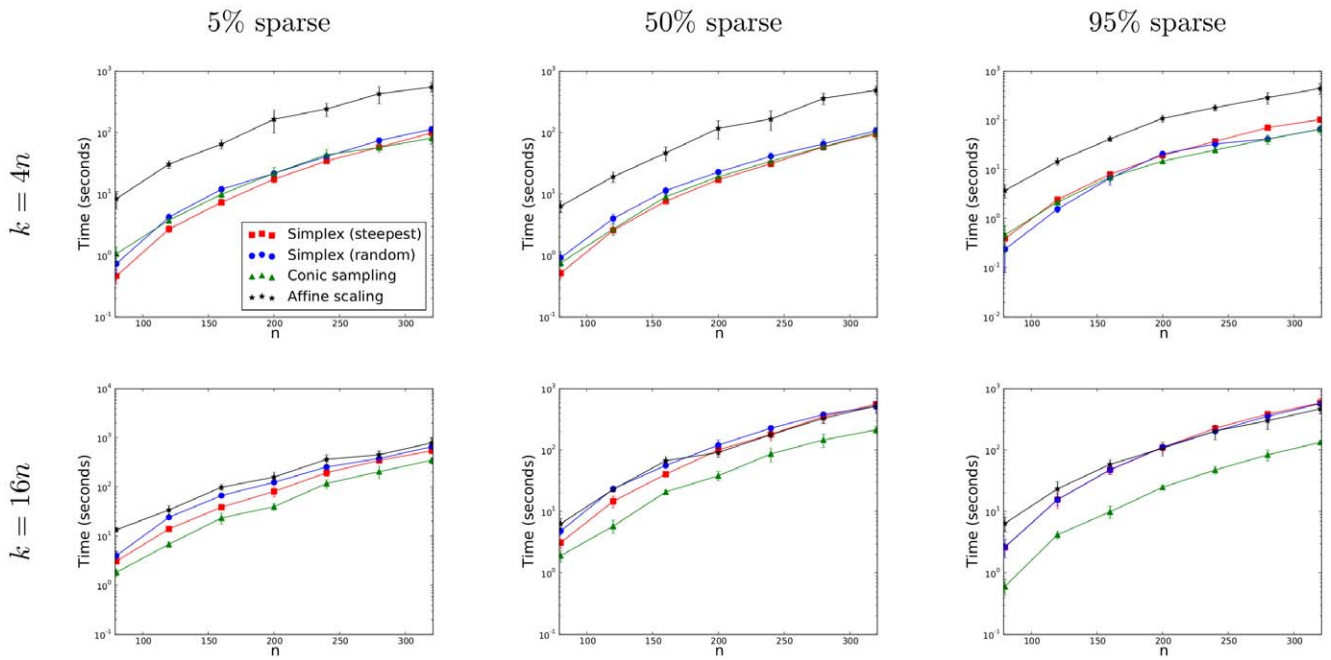

Figure 2. Preliminary runtime analysis on random LPs. The simplex and affine-scaling algorithms are timed against the conic sampling algorithm on random LP problems of varying dimension n and number of constraints $k = 2n$ and $16n$. For each n and k, five LP problems were generated. Each panel figure plots the mean runtime as a function of . Error bars indicate the minimum and maximum runtimes. Dual sparse-dense vector data structures were used.
doi:10.1371/journal.pone.0043706.g002

Figure 2. Preliminary runtime analysis on random LPs. The simplex and affine-scaling algorithms are timed against the conic sampling algorithm on random LP problems of varying dimension n and number of constraints $k = 2n$ and $16n$. For each n and k, five LP problems were generated. Each panel figure plots the mean runtime as a function of n. Error bars indicate the minimum and maximum runtimes. Dual sparse-dense vector data structures were used.

Runtimes on Highly Sparse, Random LPs with Varying Numbers of Constraints

Figure 2 illustrates that the greatest performance benefit is achieved on problems with many constraints and a high degree of sparsity. For this reason, the runtime of the conic sampling algorithm was compared to the simplex and primal affine-scaling algorithms on highly sparse (*i.e.* with 95% zero values) polytopes with $n = 100$ (chosen small enough so that it is practical for the number of constraints k to dwarf n) and variable numbers of constraints $k = 128, \ldots, 16384$. On these LPs, fully sparse vector data structures were used (the dual sparse-dense vector data structure used far too much memory).

Figure 3 shows the improvement of the conic sampling method over the affine-scaling and simplex methods on highly constrained, sparse problems.

Application to QPs from Computational Proteomics

Lastly we demonstrate the efficiency of the modified conic sampling method (*i.e.* the slight modification described above which adapts the method for computing projections on polytopes). We apply this method to polytope projections taken from computational biology. The particular QP is used to efficiently estimate protein confidences (or quantities) from the directly measured peptide confidences (or abundances) [22]. The problem has previously been modeled as an NP-hard set cover problem [23], which weights proteins from a bipartite graph of proteins and peptides observed in a mass spectrometry experiment (with edge set denoted E). The previous method finds the smallest set of proteins that explain a certain amount of observed peptide evidence. Enforcing economy in the cardinality of the protein set prevents shared peptides, which may have come from several proteins, from incorrectly resulting in multiple protein identifications.

The QP relaxation minimizes the $L2$-norm of the protein identifications (denoted x); like the set cover formulation, this QP formulation enforces economy in the protein set. Also similar to the set cover formulation is the constraint requiring a certain quantity of peptides (weighted by their scores from the mass spectrometry experiment, each denoted s_j) to be "explained." The weighted number of explained peptides is given by the hyperparameter τ. Each peptide (denoted y_j) is further constrained to equal the sum of the proteins containing it (*i.e.* the sum of proteins adjacent to the peptide in the bipartite graph). Lastly, proteins are constrained to have nonnegative scores. The final QP is a projection of the zero vector onto a polytope:

$$\text{Minimize } \|x\|_2^2 :$$

$$\forall i, x_i \geq 0$$

.

$$\forall j, y_j = \sum_{i:(i,j)\in E} x_i$$

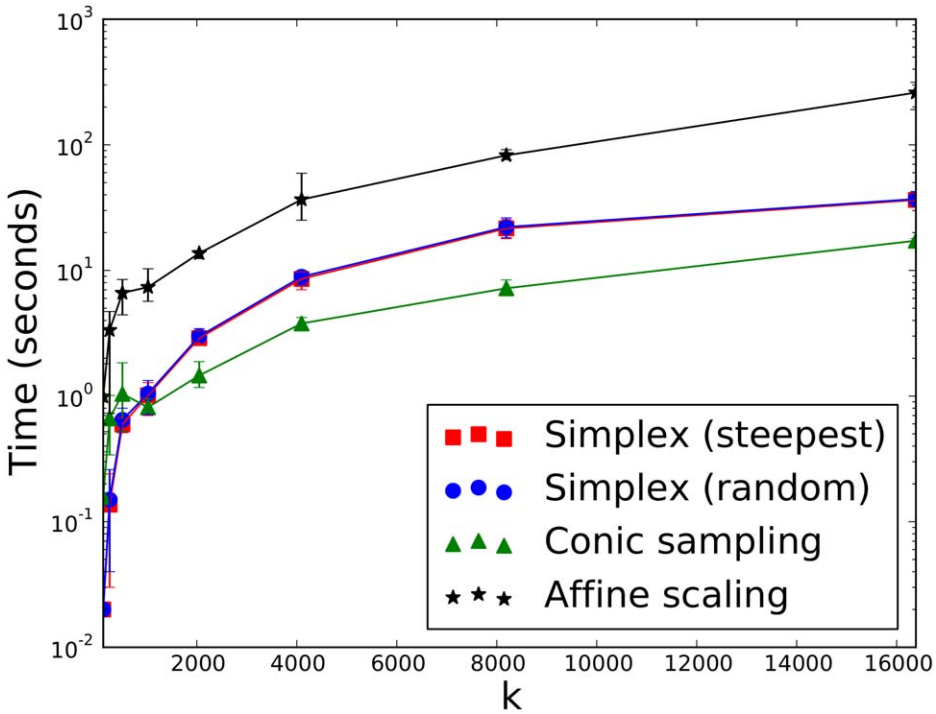

Figure 3. Runtimes on highly sparse, random LPs with varying numbers of constraints. The simplex and affine-scaling algorithms were timed against the conic sampling algorithm on random LP problems with $n = 100$, with 95% sparsity, and with number of constraints $k = 128,\ldots,16384$. For each k, three problems were generated and timed with all algorithms. Error bars indicate minimum and maximum runtimes. Fully sparse vector data structures were used.
doi:10.1371/journal.pone.0043706.g003

$$\sum_j s_j y_j \geq \tau$$

An initial feasible point was found by setting all x_i to a large value (in this case 1, because the peptide scores s_j are approximate probabilities); this point was provided as a starting point for all agorithms. The hyperparameter was chosen as 80% of all observed peptide scores $\tau = 0.8 \sum_j s_j$.

Graphs were produced from two previously described mass spectrometry data sets [24,25]: The first analyzes a small mixture of 48 purified proteins (plus common contaminants) searched against the human proteome (and a reversed database) as previously described. The second data set was aquired from yeast lysate.

Runtimes for optimizing the resulting QP was analyzed using the conic sampling method and the proprietary software package Mathematica (Table 0). Conic sampling runtimes were taken using UNIX user time, and Mathematica runtimes were taking using the built-in Timing command (so that the time to serialize the data into a native Mathematica format was not counted), and with no other processes running. In Mathematica the FindMinimum local optimization routine (which uses sparse vectors) was used, because the objective function is convex.

Discussion

In the demonstrated examples, conic sampling algorithm performs comparably to or outperforms the primal affine-scaling and simplex algorithms. Particularly noticeable is the efficiency when applied to problems with many constraints, especially those with a great deal of sparsity. The efficiency of conic sampling on highly constrained LPs makes intuitive sense; each iteration of the conic sampling algorithm is $\mathcal{O}(n^3)$ (before accounting for sparsity), and so the cost is roughly equivalent to n pivots with the simplex method (at a cost of $\mathcal{O}(n^2)$ per pivot). When the number of expected pivots exceeds n, conic sampling may be much more efficient. Likewise, in the worst case (for a completely degenerate problem), each iteration of conic sampling will advance along an edge in a manner similar to Bland's pivot rule. If this were the case, it would behave as an n-fold slower version of the simplex

Table 1. QP runtimes from computational proteomics.

	Sigma 48		Yeast lysate	
Variables (n)	392		3733	
Constraints (κ)	393		3734	
Method	**Conic sampling**	**Mathematica**	**Conic sampling**	**Mathematica**
	20.7942	20.7942	351.24	351.24
Runtime	**0.012**	1.41	**0.352**	145.68

The runtimes of Mathematica and the conic sampling algorithm (modified to compute the projection onto a polytope) are shown on QPs taken from computational proteomics (faster times are written in bold). Fully sparse vector data structures were used by conic sampling (sparse vectors are also used internally by Mathematica). The final objective value is presented using the default precision reported by Mathematica (both algorithms compute the same result).
doi:10.1371/journal.pone.0043706.t001

method. It should be noted that this n-fold cost can be recovered by only *removing* the departed constraints from the Gram-Schmidt matrix (rather than rebuilding it from scratch); however, this can decrease the numeric stability of the algorithm by allowing errors to accumulate instead of resetting each time fixation is reached. Sparse Sherman-Morrison updating or sparse LU updating can achieve the same effect, but with increased numeric precision.

Furthermore, in LPs where the number of vertices is substantially higher than the dimensionality, the length of a greedy path along adjacent vertices (*i.e.* a path taken by the simplex methods) may become very large. On such highly constrained polytopes, a ray within the polyhedral cone defined by the objective vector and the bounding constraints has a high probability of arriving at a substantially improved vertex. If the improved vertices are sampled in a roughly uniform manner, then the expected number of iterations required by conic sampling will be logarithmic in the number of vertices. It may be possible to perform a more intelligent sampling of rays in the polyhedral cone (*i.e.* importance sampling), and yield a guaranteed expected runtime for certain families of polytopes. In this manner, conic sampling can be thought of as a generalization of simplex methods: rather than choosing an edge from a finite collection, the algorithm chooses a vector in a potentially infinite collection. Rules for choosing a vector from the polyhedral cone correspond to generalizations of simplex pivot rules.

Due to its simple geometric nature, the proposed method can be modified and applied to problems other than LP. On the family of QPs that compute projections onto polytopes, the conic sampling method is significantly more efficient than a sophisticated proprietary software package (Table 1); no doubt, a superior implementation (*e.g.* using the linear algebra library from Mathematica, Matlab, or LINPACK) could possibly be faster still, and could be applied to large, sparse projections, as well as to other QPs and convex optimization problems.

The conic sampling algorithm, like several other existing methods, is motivated by a straightforward geometric notion; however, it does not suffer the same practical inefficiencies observed in the randomized simplex method. Randomness in LP solvers is typically used to avoid pathological behavior and is not responsible for good performance in practice. For conic sampling, the possibility of randomly jumping to a much improved vertex, and thus substantially narrowing the remaining vertices, is tightly intertwined with the algorithm's performance. It should be noted that there may be deterministic variants of conic sampling that optimize over vectors in the polyhedral cone rather than choosing a random vector, and that these methods may still be faster in practice than the random approach.

The conic sampling method bears a resemblence to the gravity descent method. Both methods descend along facets and edges, permitting far more direct paths to the optimum; however, the gravity descent method solves a QP in order to decide which sets of constraints to abandon. In contrast, the conic sampling method takes a more restrictive, and less computationally expensive descent by projecting along the constraints of the polytope. This descent is not guaranteed to be the steepest, and will necessarily become immobile in at most n steps. Randomly sampling from a feasible ray not only allows the conic sampling method to become "unstuck" from the bounding constraints (without solving a QP), the manner with which it does so permits large jumps that move back through the strict interior, to a distal region of the convex hull.

The conic sampling method also benefits from largely ignoring unimportant constraints in a manner reminiscient of Seidel's algorithm and the Matousek/Sharir/Welzl algorithm; however, rather than directly sifting through the constraints and finding those that bound the optimum, the conic sampling method implicitly does so. On polytopes with many facets not bounding the optimum, the chances of visiting these extraneous facets are very small, and once the objective function passes the greatest attainable on a feasible vertex of a facet, the facet is no longer considered.

The ease with which some geometrically motivated methods can be adapted to completely different convex optimization problems underscores its flexibility and generality. QPs and general convex optimization problems arise frequently, and form a superset of LPs. The greater mathematical complexity of these more general problems has resulted in fewer applicable algorithms and a greater difficulty in their optimization. Extensions of geometrically motivated methods like conic sampling algorithm may be of great use in solving these more general problems, especially in cases where the number of constraints dwarfs the dimensionality of the problem. Performing more intelligent random sampling in this cone may permit uniform sampling of the remaining volume or feasible vertices, and lead to algorithms with expected runtime bounds that are subexponential in the number of dimensions or constraints.

Acknowledgments

I am grateful for the contributions of Jeff Bilmes and William Noble, without whom this work would not have been possible. This work is dedicated to the memory of Paul Tseng, who generously shared his expertise.

Author Contributions

Conceived and designed the experiments: OS. Performed the experiments: OS. Analyzed the data: OS. Contributed reagents/materials/analysis tools: OS. Wrote the paper: OS.

References

1. Glocker B, Komodakis N, Tziritas G, Navab N, Paragios N (2008) Dense image registration through MRFs and efficient linear programming. Medical Image Analysis 12: 731–741.
2. Tsuda K, Ratsch G (2004) Image reconstruction by linear programming. In: NIPS, Cambridge, MA: MIT Press.
3. Hauskrecht M, Kveton B (2004) Linear program approximations for factored continuous-state Markov decision processes. In: NIPS, Cambridge, MA: MIT Press.
4. Yanover C, Meltzer T, Weiss Y (2006) Linear programming relaxations and belief propagation – an empirical study. J Mach Learn Res 7: 1887–1907.
5. Sanghavi S, Malioutov D, Willsky A (2008) Linear programming analysis of loopy belief propagation for weighted matching. In: NIPS, Cambridge, MA: MIT Press. 1273–1280.
6. Vanderbei RJ (1996) Linear Programming: Foundations and Extensions.
7. Dantzig GB (1947) Linear Programming and Extensions. Princeton University Press.
8. Klee V, Minty GJ (1972) How good is the simplex algorithm? In: Shisha O, editor, Inequalities III, Academic Press. 159–175.
9. Kalai G (1991) A subexponential randomized simplex algorithm. Unpublished manuscript, Institute of Mathematics, Hebrew University of Jerusalem, Jerusalem, Israel.
10. Murty KG (1988) Linear Complementarity: Linear and Nonlinear Programming.
11. Khachiyan LG (1979) A polynomial algorithm for linear programming. Soviet Math Dokl 20: 191–194.
12. Karmarkar N (1984) A new polynomial-time algorithm for linear programming. Combinatorica 4: 373–395.
13. Vanderbei R, Meketon M, Freedman B (1986) A modification of Karmarkar's linear programming algorithm. Algorithmica 1: 395–407.
14. Wright SJ (1997) Interior-Point Methods. SIAM.
15. Barnes ER (1986) A variation of Karmarkar's algorithm for solving linear programming problems. Mathematical Programming 36: 174–182.

16. Kalai G (1985) Some computational experience and a modification of the karmarkar algorithm. The Pennsylvania State University, ISME Working Paper, 85–105.
17. Y RS (1991). Small-dimensional linear programming and convex hulls made easy.
18. Matousek J, Sharir M, Welzl E (1996) A subexponential bound for linear programming. Algorithmica 16: 498–516.
19. Bland RG (1977) New finite pivoting rules for the simplex method. Mathematics of Operations Research 2: 103–107.
20. Forrest JJH, Tomlin JA (1972) Updated triangular factors of the basis to maintain sparsity in the product form simplex method. Mathematical Programming 2: 263–278.
21. Reid JK (1982) A sparsity-exploiting variant of the Bartels – Golub decomposition for linear programming bases. Mathematical Programming 24: 55–69.
22. Serang O, Noble WS (2012) A review of statistical methods for protein identification using tandem mass spectrometry. Statistics and Its Interface 5: 3–20.
23. Zhang B, Chambers MC, Tabb DL (2007) Proteomic parsimony through bipartite graph analysis improves accuracy and transparency 6: 3549–3557.
24. Serang O, MacCoss MJ, Noble WS (2010) Efficient marginalization to compute protein posterior probabilities from shotgun mass spectrometry data 9: 5346–5357.
25. Käll L, Canterbury J, Weston J, Noble WS, MacCoss MJ (2007) A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets 4: 923–25.