Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

DESIGNING A FRANK-WOLFE ALGORITHM FOR SIMULATION OPTIMIZATION OVER UNBOUNDED LINEARLY CONSTRAINED FEASIBLE REGIONS

Natthawut Boonsiriphatthanajaroen[1], and Shane G. Henderson[1]

[1]School of Operations Research and Information Engr., Cornell University, Ithaca, NY, USA

ABSTRACT

The linearly constrained simulation optimization problem entails optimizing an objective function that is evaluated, approximately, through stochastic simulation, where the finite-dimensional decision variables lie in a feasible region defined by known, deterministic linear constraints. We assume the availability of unbiased gradient estimates. When the feasible region is bounded, existing algorithms are highly effective. We attempt to extend existing algorithms to also allow for unbounded feasible regions. We extend both the away-step (AFW) and boosted Frank-Wolfe (BFW) algorithms. Computational experiments compare these algorithms with projected gradient descent (PGD). An extension of BFW performs the best in our experiments overall, performing substantially better than both PGD and AFW. Moreover, PGD substantially outperforms AFW. We provide commentary on our experimental results and suggest avenues for further algorithm development. The article also showcases the use of the SimOpt Library in algorithm development.

1 INTRODUCTION

The problem we consider is to

$$\min_{x\in\mathscr{X}} f(x) = \mathbb{E}f(x,\xi) \tag{1}$$

where the non-empty feasible region $\mathscr{X}$ is the linearly constrained region $\{x \in \mathbb{R}^p : Ax \le b\}$, the matrix A and vector b are deterministic and known, ξ is some random element, the distribution of which does not depend on x, and the function $f(x,\cdot)$ is real-valued and integrable for all feasible x. We assume that the function $f(x)$ can only be observed through realizations of $f(x,\xi)$ obtained from a simulation oracle that generates i.i.d. replications of ξ and then evaluates $f(x,\xi)$. This problem is then a *linearly constrained simulation optimization problem* (LCSOP). We further assume that we have sufficient regularity so that the function $f(\cdot,\xi)$ is differentiable in its first argument, x, for all realizations of ξ, and that the (pathwise) derivative $\nabla f(x,\xi)$ of $f(\cdot,\xi)$ at the point x is integrable and unbiased as an estimator of $\nabla f(x)$, for all feasible x. We further assume that we can observe i.i.d. realizations $\nabla f(x,\xi_i)$ based on i.i.d. realizations $i = 0,1,2,\ldots$ of ξ.

LCSOPs are important. For example, Cornell University solved a problem during the COVID-19 pandemic that determined surveillance testing capacity for different campus groups (Frazier et al. 2022). There the goal was to select testing rates for the groups so as to minimize pandemic outcomes like infections over a full semester. A single linear constraint represented the bounded testing capacity. The pandemic outcomes were estimated through a stochastic simulation. This problem can be formulated as a LCSOP. If one relaxes the bound on testing capacity and instead adds a cost term to the objective function for high testing rates, the problem becomes a LCSOP with an unbounded feasible region. This latter formulation is closer to what happened in practice, where an initial bound on testing capacity was increased when the potential epidemic prevention benefits of more testing were better understood.

A broad class of algorithms can be brought to bear on LCSOPs. For example, Boonsiriphatthanajaroen et al. (2024) explored projected gradient descent (PGD), an away-step Frank-Wolfe (AFW) algorithm and an active set method in combination with 4 different line search strategies through computational comparisons on LCSOPs. There it was found that PGD and the Frank-Wolfe algorithm typically outperformed the active set method, though not universally. One of the problems explored in that paper involved an unbounded feasible region, but the variant of the Frank-Wolfe algorithm that was employed required a *bounded* feasible region. To handle this difficulty, a linear constraint was added that rendered the feasible region bounded and that was sufficiently "loose" that it was not expected to change the optimal solution to the problem. While this strategy allowed the Frank-Wolfe algorithm to make progress, the algorithm struggled on this problem compared to the other algorithms.

Given the strong overall performance of the Frank-Wolfe algorithm in Boonsiriphatthanajaroen et al. (2024), except on an unbounded problem, we are motivated to seek an improved Frank-Wolfe algorithm for LCSOPs that requires no tailoring when encountering a problem with an unbounded feasible region. It is important that no tailoring be required, to minimize the knowledge required of a user. We want to retain the strong performance of the Frank-Wolfe algorithm on bounded problems, while also ensuring competitive performance on unbounded problems. (Problems with unbounded feasible regions are common in practice, e.g., in inventory problems with unbounded order quantities, or in capacity expansion problems where capacity comes at a cost rather than being constrained.) Wang et al. (2022) develop a Frank-Wolfe method for linearly constrained unbounded regions, but their method applies only to very specially structured feasible regions that are a direct sum of a linear subspace and a bounded constraint set.

In this paper we attempt to develop a Frank-Wolfe algorithm for *general* LCSOPs. The new algorithms we explore are not a direct extension of the standard Frank-Wolfe algorithm (Frank and Wolfe 1956). Rather, we first choose to extend the AFW algorithm (Wolfe 1970, §8) and, separately, the boosted Frank-Wolfe (BFW) algorithm (Combettes and Pokutta 2020). These algorithms have demonstrated strong performance on bounded feasible regions and are thus natural candidates for extension. Our first extensions allow these algorithms to incorporate not just *vertices* of the feasible region in specifying the search direction, but also *extreme directions*. Further details are discussed in Section 2.

In Section 4 we compare the empirical performance of the new algorithms, along with projected gradient descent, on a suite of test problems that consists of a mix of bounded and unbounded LCSOPs. The test problems are briefly described in Section 3. We elect not to include active-set methods in the comparison because their performance lagged the others that were explored in Boonsiriphatthanajaroen et al. (2024).

The results in Section 4 indicate some promise, but there is a repeated problem, that we term "plateauing," on problems with an unbounded feasible region where the algorithms fail to make much progress for many iterations. Investigation of the results suggests that plateauing arises because the algorithms are heading in an unbounded direction, i.e., a direction in which the feasible region is unbounded, but the unbounded direction is poorly aligned with the negative gradient.

Accordingly, in Section 5 we provide some algorithm adjustments that, we hope, address the plateauing effect. Some of these new algorithms are motivated by cutting-plane algorithms for convex optimization problems, in that they entail adding constraints based on an interaction between the convex structure of the objective function and the (estimated) gradients. Empirical results for all of the proposed algorithms are given in Section 6, and, in Section 7, we discuss what we learn from the experiments and conclude.

We view the main contributions of the paper to be the design of the new Frank-Wolfe algorithm variants and their empirical comparisons that shed light on their performance relative to selected competing algorithms.

2 ALGORITHMS

Let x_k be the incumbent solution in the kth iteration. Let $\hat{f}$ denote a simulation estimator of the function f, e.g., the usual sample-average approximation $\hat{f}(x) = n^{-1}\sum_{i=1}^{n} f(x;\xi_i)$ with $(\xi_1, \xi_2, \ldots)$ denoting an iid sequence where each ξ_i is equal in distribution to the random element ξ. Let g_k be the estimated gradient at x_k, which, in the setting where $f(\cdot,\xi)$ is appropriately differentiable and some extra regularity holds, could be taken to be the infinitesimal perturbation analysis estimator $n^{-1}\sum_{i=1}^{n} f'(x_k,\xi_i)$. The algorithms described below need a line search method to determine step sizes, for which we use interpolation, which performed well in Boonsiriphatthanajaroen et al. (2024).

2.1 Projected Gradient Descent

Algorithm 1, Projected Gradient Descent (PGD), adapted from Iusem (2003), moves in the direction of the negative gradient, uses an L_2 projection $\mathscr{P}_{\mathscr{X}}$ back to the feasible region to retain feasibility, then performs a linesearch in the direction of the projected point. The projection is obtained by solving a quadratic program using CVXPY (Diamond and Boyd 2016, Agrawal et al. 2018). Stepsizes are constrained to an iteration-specific maximum stepsize that is chosen adaptively.

Algorithm 1 Projected gradient descent (PGD).

```
Initialize feasible solution $x_0 \in \mathscr{X}$; maximum step size $\gamma_0^{max} \in [1,\infty)$; step size discount factor $r \in (0,1)$
for $k = 0,1,2,...$ do
    $d_k \leftarrow -g_k$
    $y_{k+1} \leftarrow x_k + \gamma_k^{max} d_k$
    if $y_{k+1}$ is NOT feasible then
        $y_{k+1} \leftarrow \mathscr{P}_{\mathscr{X}}(y_{k+1})$
        $d_k \leftarrow \frac{(y_{k+1} - x_k)}{\gamma_k^{max}}$
    end if
    Let $\gamma_k = \arg\min_{\gamma \in [0,\gamma_k^{max}]} \hat{f}(x_k + \gamma d_k)$    (line search)
    if $\gamma_k = \gamma_k^{max}$ then
        $\gamma_k^{max} \leftarrow \frac{\gamma_k^{max}}{r}$
    else
        $\gamma_k^{max} \leftarrow r\gamma_k^{max}$
    end if
    $x_{k+1} \leftarrow x_k + \gamma_k d_k$
end for
```

2.2 Away-Step Frank-Wolfe

The original Frank-Wolfe algorithm (Frank and Wolfe 1956) iteratively moves towards a vertex of the feasible region that is obtained by solving a linear program that minimizes a linearization of the objective function. This can result in zigzagging behavior, which is inefficient; see Figure 1. Algorithm 2, the AFW algorithm (Wolfe 1970; Julien and Jaggi 2015), additionally allows a step in a direction that moves away from a "bad" vertex. A vertex representation of the current solution is maintained as a convex combination of a subset of the vertices $\mathscr{V}$ of the feasible region, which is standard, plus a conic combination of the extreme directions of the feasible region, $\mathscr{W}$, which is new. More precisely, we write $x_k = \sum_{v\in\mathscr{V}} \alpha_v^{(k)} v + \sum_{w\in\mathscr{W}} \beta_w^{(k)} w$, where all coefficients are non-negative and $\sum_{v\in\mathscr{V}} \alpha_v^{(k)} = 1$. Define the *active vertex set* at iteration k, $\mathscr{S}^{(k)}$, to be all vertices (not extreme directions) with positive coefficients, i.e. $\mathscr{S}^{(k)} = \{v \in \mathscr{V} | \alpha_v^{(k)} > 0\}$. We initialize our solution x_0 to be a vertex $v_0 \in \mathscr{V}$, so we initialize all coefficients to be zero except v_0 which has coefficient 1.

Vertices and extreme directions are obtained by solving a linear program. Let LP(r) be a function that returns either a vertex solution to the linear program $\max_{x\in\mathscr{X}} \langle r,x\rangle$ when the solution is finite, or an extreme direction u of unit norm such that $\langle r,u\rangle > 0$ and such that $x + \lambda u$ is feasible for any feasible x and $\lambda > 0$.

In each iteration, we first solve LP($-g_k$), i.e., we solve a linear program whose objective function is the linearized function. If the optimal solution is a vertex, s_k, then the potential search direction is $d_k^{FW} = s_k - x_k$, i.e., we search in the direction of that vertex. If, instead, the linear program is unbounded and returns the (normalized) extreme direction s_k, then the potential search direction is $d_k^{FW} = s_k$. Separately, we also compute the worst active vertex, v_k, meaning that v_k maximizes $\langle g_k, v\rangle$ over all active vertices, and then define the potential search direction $d_k^A = x_k - v_k$. We then select either d_k^{FW} or d_k^A as the search direction, whichever maximizes a certain dot product.

The key difference with the usual AFW algorithm for bounded feasible regions is that the linear program can be unbounded. In that case, we use the extreme direction that is returned by the linear program solver as a search direction. The "bookkeeping" for maintaining the representation of the current point as not only a convex combination of extreme points, which is standard, but also a conic combination of extreme directions, which is new, is also slightly more complicated.

Depending on which kind of step we take, we have different limits on the maximum possible step size. For a normal FW step towards a vertex of the form $d_k^{FW} = s_k - x_k$, the maximum step size is 1, corresponding to a move to the target vertex s_k. For an away step, we have to limit the maximal possible step size so that the weight on the worst active vertex, v_k, remains positive. The update to the current solution is $x_{k+1} = x_k + \gamma_k(x_k - v_k)$, which implies that the weight on the worst vertex, v_k, changes from its current value, $\alpha_{v_k}^{(k)}$, to $(1+\gamma_k)\alpha_{v_k}^{(k)} - \gamma_k$. For this to remain nonnegative, we require that the step size γ_k be at most $\alpha_{v_k}^{(k)}/(1-\alpha_{v_k}^{(k)})$.

In the case that the search direction is the extreme direction, $d_k^{FW} = s_k$, obtained by solving the linear program, the step size is more complicated. We retain a parameter, γ_k^{max} that limits the maximum possible step size in Iteration k. In each iteration of this form we perform a line search for the step size $\gamma_k \in [0, \gamma_k^{max}]$. If the optimal step size, γ_k is attained at the endpoint, γ_k^{max}, then we increase the maximal step size by a (geometric) growth factor, i.e., we set $\gamma_{k+1}^{max} = \gamma_k^{max}/r$ for some $r < 1$. If, instead, the optimal step size, γ_k, is interior to the interval then we reduce the maximal step size to γ_k.

At the end of each iteration, we update the vertex representation of x_k as follows:

1. Normal FW Step with search direction $s_k - x_k$:
    - If $\gamma_k = \gamma_k^{max} = 1$, then set $\mathscr{S}^{(k+1)} = \{s_k\}$
    - If $\gamma_k < \gamma_k^{max} = 1$, then set $\mathscr{S}^{(k+1)} = \mathscr{S}^{(k)} \cup \{s_k\}$
    - Update the weights $\alpha_{s_k}^{(k+1)} = (1-\gamma_k)\alpha_{s_k}^{(k)} + \gamma_k$
    - Update the weights $\alpha_v^{(k+1)} = (1-\gamma_k)\alpha_v^{(k)}$ for all $v \in \mathscr{S}^{(k)} \setminus \{s_k\}$
    - Update the weights $\beta_w^{(k+1)} = (1-\gamma_k)\beta_w^{(k)}$ for all $w \in \mathscr{W}$
2. Away step with search direction $x_k - v_k$:
    - If $\gamma_k = \gamma_k^{max}$, then set $\mathscr{S}^{(k+1)} = \mathscr{S}^{(k)} \setminus \{v_k\}$
    - If $\gamma_k < \gamma_k^{max}$, then set $\mathscr{S}^{(k+1)} = \mathscr{S}^{(k)}$
    - Update the weights $\alpha_{v_k}^{(k+1)} = (1+\gamma_k)\alpha_{v_k}^{(k)} - \gamma_k$
    - Update the weight $\alpha_v^{(k+1)} = (1+\gamma_k)\alpha_v^{(k)}$ for all $v \in \mathscr{S}^{(k)} \setminus \{v_k\}$.
    - Update the weights $\beta_w^{(k+1)} = (1+\gamma_k)\beta_w^{(k)}$ for all $w \in \mathscr{W}$
3. Extreme direction with search direction s_k:
    - Update the weight $\beta_{s_k}^{(k+1)} = \beta_{s_k}^{(k)} + \gamma_k$ for the unbounded direction s_k

2.3 Boosted Frank-Wolfe

The AFW algorithm (Algorithm 2) was introduced to mitigate the zig-zagging behavior of the standard Frank-Wolfe algorithm. However, its search direction can still be poorer than that of gradient descent. Algorithm 3, the BFW algorithm, is adapted from Combettes and Pokutta (2020) to allow an unbounded feasible region. Within each iteration, k, it attempts to represent the negative gradient as a conic combination of the vertices, $\mathscr{V}$, as in the original work, but also, in our setting, the extreme directions of the feasible region. This approach implicitly addresses the optimization subproblem

$$\min_d \|-g_k - d\|^2, \tag{2}$$

Algorithm 2 Away-step Frank-Wolfe (AFW).

```
Initialize $x_0$ to be a vertex $v$, set $\alpha_v^{(0)} = 1$ and all other coefficients to 0, set a step size discount factor $r \in (0,1)$
for $k = 0,1,2,\dots$ do
    $s_k \leftarrow \text{LP}(-g_k)$; the usual Frank-Wolfe vertex or an extreme direction
    $v_k \leftarrow \arg\max_{v \in \mathscr{S}^{(k)}} \langle g_k, v \rangle$; a "bad" vertex
    $d_k^{FW} \leftarrow \begin{cases} s_k - x_k & \text{if the LP solution was a vertex} \\ s_k & \text{if the LP solution was an unbounded direction} \end{cases}$
    $d_k^A \leftarrow x_k - v_k$   (away direction)
    if $-g_k^T d_k^{FW} \geq -g_k^T d_k^A$ then
        $d_k \leftarrow d_k^{FW}$
        $\gamma_k \leftarrow \begin{cases} \arg\min_{\gamma \in [0,1]} \hat{f}(x_k + \gamma d_k) & \text{if LP gives a vertex solution} \\ \arg\min_{\gamma \in [0, \gamma_k^{max}]} \hat{f}(x_k + \gamma d_k) & \text{if LP gives an unbounded direction} \end{cases}$
        $\gamma_{k+1}^{max} \leftarrow \begin{cases} \frac{\gamma_k^{max}}{r} & \text{if LP gives an unbounded direction and } \gamma_k = \gamma_k^{max} \\ \gamma_k & \text{if LP gives an unbounded direction and } \gamma_k < \gamma_k^{max} \end{cases}$
    else
        $d_k \leftarrow d_k^A$
        $\gamma_k \leftarrow \arg\min_{\gamma \in [0, \alpha_{v_k}^{(k)}/(1-\alpha_{v_k}^{(k)})]} \hat{f}(x_k + \gamma d_k)$   (line search)
    end if
    $x_{k+1} \leftarrow x_k + \gamma_k d_k$
    Update the weights $\alpha^{(k)}$ giving the vertex representation of $x_{k+1}$ to $\alpha^{(k+1)}$
end for
```

where g_k is the estimated gradient of the solution x_k at iteration k. The domain over which we explore consists of the conic combinations of $\mathscr{V} - x_k$, which is standard, and the extreme directions of $\mathscr{X}$, which is new.

Within each iteration, the algorithm successively refines the direction d by attempting to improve its alignment with the negative estimated gradient, $-g_k$, by projecting onto a one-dimensional line using the Non-Negative Matching Pursuit Algorithm (Locatello et al. 2017); see Figure 2. Alignment is measured by the function *align*, defined as

$$align(u,v) = \begin{cases} \frac{u^T v}{||u|| \cdot ||v||} & \text{if } u, v \neq 0, \\ -1 & \text{if } u = 0 \text{ or } v = 0, \end{cases}$$

which is, essentially, the cosine of the angle between the vectors u and v. The algorithm successively improves the alignment, in each iteration solving a linear program that identifies the vertex or the extreme direction that best can be used to reduce the difference between the negative gradient and the current search direction, stopping only when it fails to improve the alignment by at least some minimal threshold, δ, which is typically set to 10^{-3}. In Algorithm 3, T denotes the maximum number of these alignment-improvement steps permitted in the Boosted Direction Oracle, and it is typically set to 10. When $T = 1$, BFW reduces to the standard Frank-Wolfe method. The maximum step size γ_k^{max} is determined from the minimum ratio test. The maximum step size could be infinite if the feasible region is unbounded in the search direction. In this case, we actually perform a line search over step sizes on the interval $[0,1]$. For default BFW parameter choices and line search parameters, see the implementation in the SimOpt Library (2025).

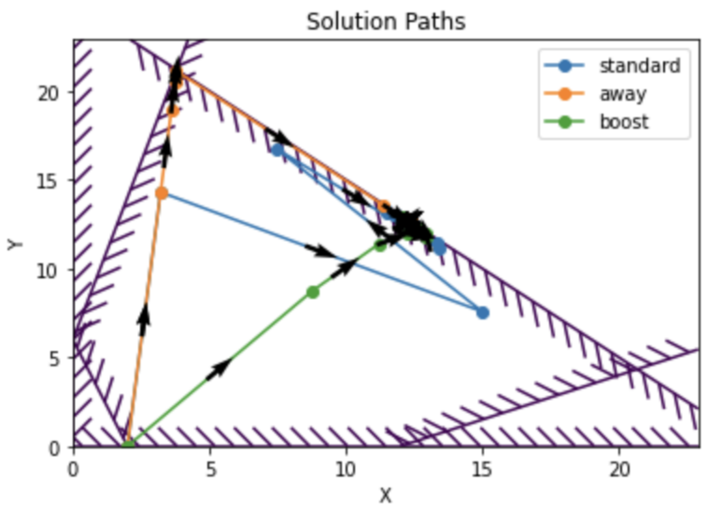

Figure 1: The AFW algorithm mitigates zigzagging behavior in the original Frank-Wolfe algorithm, by moving not just towards vertices, but also away from vertices. The BFW algorithm has even more flexibility in choosing a search direction.

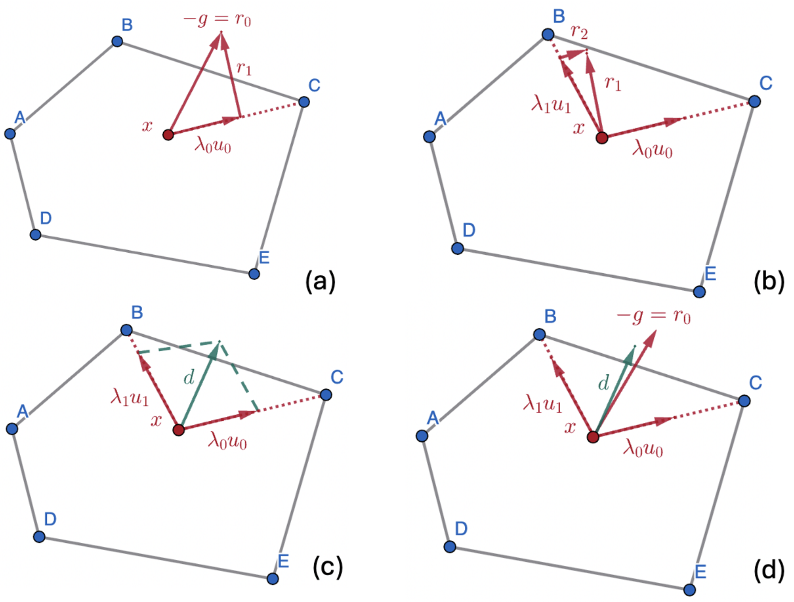

Figure 2: Computing the search direction. In (a), Vertex C solves the linear program, and we project the negative gradient onto u_0, leaving the residual r_1. In (b), Vertex B solves the linear program, and we project the residual, r_1, onto u_1, leaving the residual r_2, which sufficiently improves alignment and thus defines the direction d in (c). In (d), the alignment between the direction d and the negative gradient is highlighted.

3 PROBLEMS

Here we provide only brief remarks on the choices of test problems. For full details on the test problems, see the SimOpt Library (2025).

Our test problems consist of 10 random instances of each of 1) the Constrained Stochastic Activity Network (SAN), 2) the Stochastic Max Flow (SMFCVX) problem, and 3) the Open Jackson Network (OJN)

Algorithm 3 Boosted Frank-Wolfe (BFW).

```
Initialize $x_0$ to be a vertex $v$, maximum number of oracle steps $T$, an improvement tolerance $\delta$.
for $k = 0,1,2,...$ do
    Estimate the gradient $\nabla f(x_k)$ by $g_k$
    $d_k, \Lambda_k \leftarrow$ Boosted Direction Oracle$(T, \delta, g_k)$
    $d_k \leftarrow d_k / \Lambda_k$
    $\gamma_k^{max} \leftarrow$ maximum step size computed from minimum ratio test
    $\gamma_k \leftarrow \arg\min_{\gamma \in [0, \gamma_k^{max}]} \hat{f}(x_k + \gamma d_k)$    (line search)
    $x_{k+1} \leftarrow x_k + \gamma_k d_k$
end for
```

Algorithm 4 Boosted direction oracle.

```
Accept as input $T$, the maximum number of iterations, $\delta$, a parameter to indicate sufficient alignment,
and $g$, the estimated gradient
Initialize $d_0$ to be a zero vector for the direction, $\Lambda = 0$ to be a normalizing term.
for $t = 0,1,2,...,T-1$ do
    $r_t \leftarrow -g - d_t$; this is the running residual
    $v_t \leftarrow \mathrm{LP}(r_t)$; solve the LP
    if LP returned a vertex solution then
        $w = v_t - x$
    else
        $w = v_t$
    end if
    $u_t \leftarrow \arg\max_{u \in \{w, -d_t/\|d_t\|\}} r_t^T u$; see Locatello et al. (2017) to understand this step
    $\lambda_t \leftarrow r_t^T u_t / \|u_t\|^2$
    $d' \leftarrow d_t + \lambda_t u_t$
    if $align(-g, d') - align(-g, d_t) \geq \delta$ then
        $d_{t+1} \leftarrow d'$
        $\Lambda = \begin{cases} \Lambda + \lambda_t & \text{if } u_t = w \\ \Lambda(1 - \lambda_t/\|d_t\|) & \text{if } u_t = -d_t/\|d_t\| \end{cases}$
    else
        break
    end if
end for
return $d_t, \Lambda$
```

problem, for a total of 30 test problems. All problems are convex, and we obtain gradient estimates using infinitesimal perturbation analysis. Both SAN and OJN have unbounded feasible regions, while SMFCVX has a bounded feasible region. The SAN problem involves minimizing the maximum-length path from a source node to a sink node in a feed-forward network. The task durations (arc lengths) are random, though controlled through decision variables specifying their means. The SMFCVX problem entails maximizing the flow through a capacitated network, the capacities of which are random but partially controlled through decision variables. The OJN problem entails minimizing the mean steady-state number of jobs in an open Jackson queueing network, with decision variables giving the mean service rates.

4 INITIAL EXPERIMENTS

All of our experiments use the SimOpt Library (2025). In reporting the results, we compare the various algorithms based only on the simulation replications used; we do not compare the algorithms on their *processing time*. This is deliberate; see Eckman et al. (2023) for a detailed justification of this approach. In short, the reason is that simulation models can be computationally expensive to run compared with deterministic algorithmic elements such as computing projections in PGD or solving linear programs in the Frank-Wolfe algorithm variants.

All problems are run using common random numbers across all solvers, to ensure a fair comparison; see Eckman et al. (2023). We ensure different random problem instances are generated using different substreams, and each distinct problem factor and model factor has its own individual subsubstream. We name the problems we generated by the names introduced in Section 3 with the instance's label. For example, the problem "SAN-R10" is the 10^{th} instance of the SAN problem.

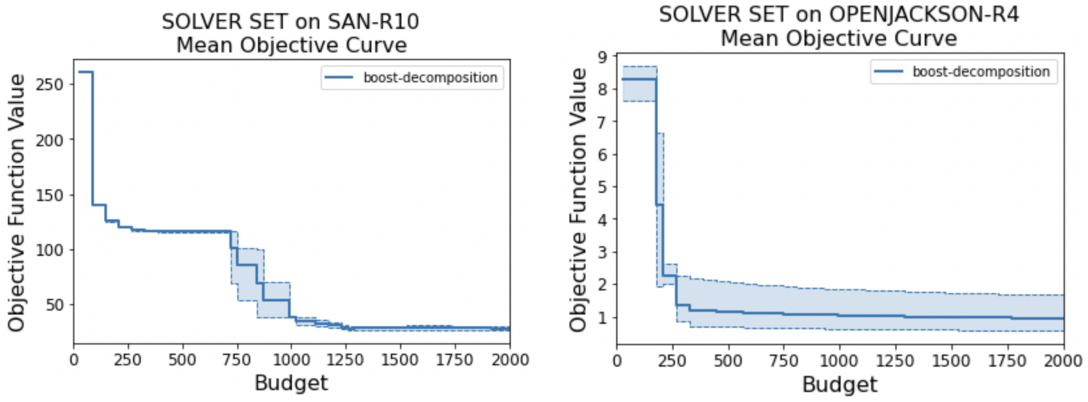

Figure 3: An instance of SAN tackled using Algorithm 3 exhibits a plateau in the budget interval $250-750$ (left), and an instance of OJN has a plateau in the budget interval $500-2000$.

Algorithm 3, which we name *Decomposition*, attempts to align the negative gradient with a conic combination of Frank-Wolfe and extreme directions. The use of a combination of vertices as well as extreme directions can improve the alignment with the negative gradient compared to the standard Frank-Wolfe algorithm. However, poor alignment is still possible when the feasible region is unbounded. Poor alignment is evident in progress plots. These plots give the estimated objective function value of the best solution found to date on the vertical axis and the expended budget, as measured in simulation replications, on the horizontal axis. Plateaus are evident in Figure 3, in which the objective function is not improved for an extended period.

5 ADDITIONAL ALGORITHMS

To mitigate the plateau effect observed in Figure 3, we propose several variants of the BFW algorithm that incorporate a few key modifications. These modifications are mostly intended to enhance the set of search directions available to the BFW algorithm, thereby better aligning the search direction to the negative gradient.

1. **Negative Gradient**:
   We compute the search direction d_k as described in Algorithm 3. If the maximum step size γ_k^{max} computed from the minimum ratio test is infinite, we recompute γ_k^{max} using the direction of the negative gradient. If this recomputed γ_k^{max} is non-zero, we replace the search direction, d_k, with the negative gradient; otherwise we keep the original d_k.

2. **Outward Cutting**: When the search direction is unbounded, we add a new constraint (cut), hoping that the cut creates new vertices. The new cut at such an iteration, k, is $\langle g_k, x\rangle \leq \langle g_k, x_k\rangle$ where x_k is the current solution at iteration k. This cut is valid if we use sample-average approximation with a fixed sample size and the gradient estimate, g_k, is obtained using infinitesimal perturbation analysis.
3. **Cutting**: This is similar to the previous variation, but a new cut is added in every iteration.
4. **Away**: In addition to directions that are towards vertices and extreme directions, we also allow the algorithm to consider away directions of the form $x_k - v_k$ where x_k is the solution at iteration k and v_k is the bad vertex that is computed in the same manner as in the AFW algorithm. In this variant, we add vertices to the active set if they are selected in the Boosted Direction Oracle and remove the "bad" vertex from the active set when its weight reaches 0.

6 PRIMARY EXPERIMENTS

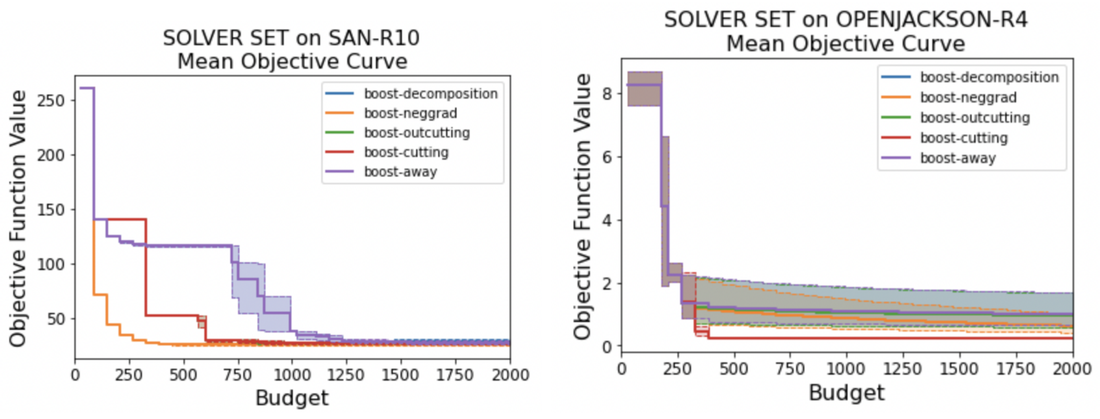

Figure 4: An instance of SAN (left), and OJN (right), tackled using variants of Algorithm 3.

Figure 4 illustrates the performance of the BFW variants on the same problems shown in Figure 3, to demonstrate that plateauing has been reduced. In the SAN-R10 problem, both the away-step and decomposition methods exhibit plateau behavior (their curves lie on top of one another), whereas the negative gradient and cutting-plane variants significantly reduce this effect. While the decomposition method tends to move steadily toward the interior of the feasible region (not shown in the plots, but observed in the experiment), the negative gradient method follows the negative gradient direction more closely, leading to improved performance. The cutting-plane methods (both outward cutting and cutting) alleviate the plateau by introducing additional constraints. These constraints generate new vertices, which enrich the set of directions available for constructing search directions. In the OPENJACKSON-R4 problem, the cutting-plane method outperforms the other variants. In this instance, *cutting* has a better performance than *outward cutting*. The additional constraints and resulting vertices from adding cuts at each step appear to enhance the search direction, accelerating convergence and helping to avoid the plateau. This improvement is particularly notable when the search direction leads to a bounded maximum step size, (not shown), allowing the algorithm to make meaningful progress. In comparison, the *decomposition* method often moves deeper into the interior without such bounded steps, resulting in slower convergence.

Figure 5 summarizes the performance of all BFW variants on all test problems. To recap, Eckman et al. (2023) where the curves in Figure 5 are introduced, the area-under-curve plot computes the mean and standard deviation of the area under the progress curve on different macroreplications, while the CDF solvability profile gives the fraction of problems solved to within 20% of the initial optimality gap. The scatter plot suggests similar overall performance among the variants. However, both the *cutting* and *outward cutting* methods exhibit higher standard deviations and significantly larger mean areas in one of the SAN problem

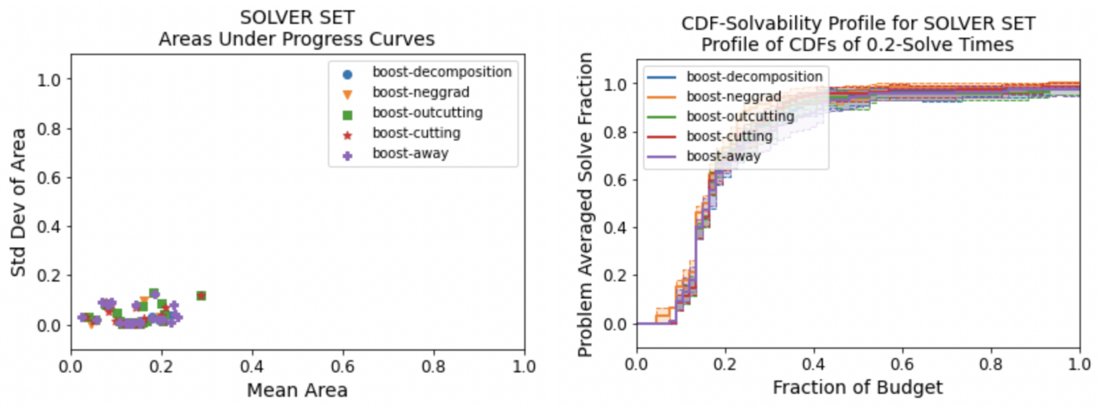

Figure 5: Area scatter plot of different variants of BFW solvers (left) and CDF solvability profile at level 0.2 (right) on all test problems.

instances. While these methods aim to enrich the search space by introducing additional constraints, the added cuts may inadvertently remove useful vertices, thereby impairing rather than enhancing the available search directions as shown in the area under progress curves where cutting-plane methods have high mean areas on one problem instance. The CDF solvability profile indicates that all variants are able to solve most problems within the given budget, with the negative gradient variant showing a slight performance advantage over the others.

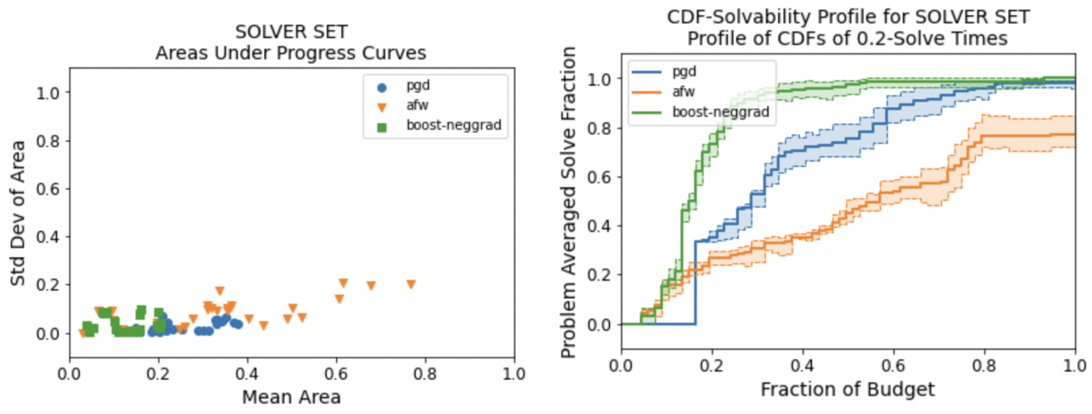

Figure 6: Area scatter plot on different solvers (left) and CDF solvability profile at level 0.2 (right).

Figure 6 compares PGD, AFW, and BFW with negative gradient. The area scatter plot indicates that BFW generally outperforms both PGD and AFW. While PGD exhibits a lower standard deviation in area, it also shows a larger mean area under the progress curve, suggesting a slower convergence toward the optimal solution. AFW, on the other hand, has both higher standard deviation and mean area than the other two algorithms, indicating more variability in performance, though it may perform well on some instances. In the solvability profile, both BFW and PGD solve nearly all instances, whereas AFW solves only approximately 75% of the problems.

Figure 7 shows the performance of the three algorithms—PGD, AFW, and BFW with negative gradient—on the SMFCVX-R5 and OPENJACKSON-R6 problems. On SMFCVX-R5, both AFW and BFW reach good-quality solutions very quickly, but eventually PGD finds a better solution. In this example, the optimal solution lies on or near a vertex, and AFW can quickly move toward it, resulting in fast convergence. In contrast, BFW prioritizes search directions that are closely aligned with the negative gradient,

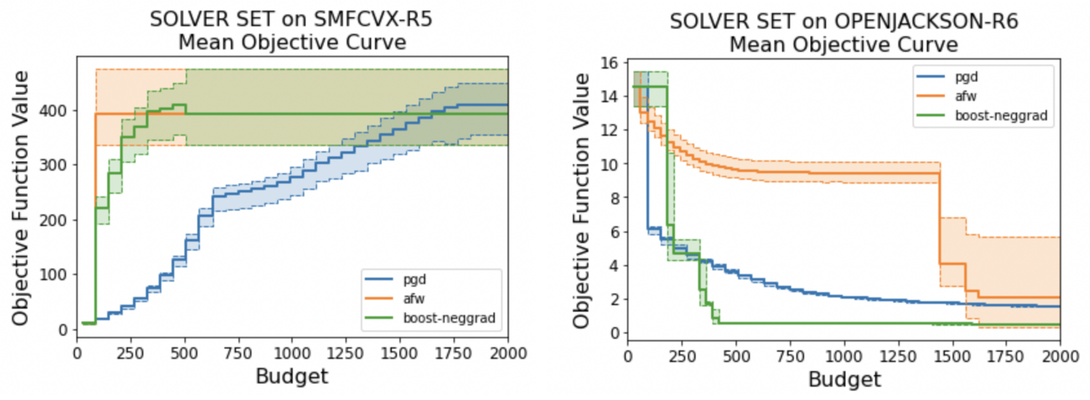

Figure 7: Mean progress curves of SMFCVX-R5 (left) and OPENJACKSON-R6 (right).

which may not point directly toward a vertex, thereby slowing its performance. Still in the SMFCVX-R5 plot, PGD exhibits two distinct phases in its progress curve: it initially moves quickly along the negative gradient with increasing step sizes, but after reaching the boundary around budget 750 its progress slows considerably. In the OPENJACKSON-R6 problem, AFW performs poorly with the unbounded feasible region, particularly in the budget range of $500-1500$, where it repeatedly selects an unbounded direction with little improvement. In contrast, PGD steadily follows the negative gradient, and BFW outperforms both alternatives.

7 DISCUSSION AND CONCLUSIONS

When the feasible region is bounded, the algorithms we developed all work relatively well. The key challenge is to also handle unbounded feasible regions. Our first algorithms exhibit a plateauing effect, in which the algorithms appear to stall for some period of time before once again making rapid progress. This plateauing effect seems to arise when searching in unbounded directions, i.e., with search directions that permit unbounded step sizes while remaining feasible.

Plateauing arises when there are limited resources for choosing a direction. The AFW algorithm exhibits a more pronounced plateauing effect because it is restricted to moving toward or away from vertices and along extreme directions. In contrast, BFW forms a combination of directions toward both vertices and extreme points, resulting in directions that more closely align with the negative gradient. This allows greater flexibility and reduces plateauing. While PGD does not exhibit this effect, similar behavior can occur when the algorithm repeatedly hits the boundary and projects back into the feasible region.

We proposed several algorithmic modifications to attempt to obviate plateauing in the BFW algorithm. All of the new variants perform better; performance is significantly improved, though still not completely satisfactory, with some residual plateauing. Of our proposed modifications, the variant of BFW that permits a search in the direction of the negative gradient performed the best in both speed in making progress towards an optimal solution and in the quality of the solution attained once the simulation budget was expended. Other variants do better in selected problem instances, but BFW with negative gradient searches is reliable and fast. Cutting plane algorithms may be worth further study, especially since they can render the feasible region bounded, but they can also limit the search direction options due to removing vertices.

Our extension of the AFW algorithm for unbounded regions has mostly disappointing performance, which arises primarily due to the plateauing effect. We have attempted to limit that effect through various algorithmic innovations, but the effect persists in experiments. We conjecture that there is an interaction between the choices of search directions and the step sizes for the line search that, with further study, might improve the performance. The algorithm does, however, perform well compared to the other algorithms when the optimal solution lies near a vertex.

Both PGD and BFW with negative gradient searches perform reasonably well, but the latter was the clear winner in our experiments.

In conclusion, we believe that our newly developed BFW algorithm (with negative gradient searches) shows strong promise, whether the feasible region is bounded or unbounded, and that we have been successful in our goal of delivering a Frank-Wolfe style algorithm that can handle unbounded feasible regions. The algorithm still exhibits some plateauing effects that merit further study. On a perhaps related note, we believe there is the potential for "scaling" issues, in which the algorithm may not adapt when we change the scale of the decision variables; the algorithm is not "scale free."

As a final comment, this article showcases the utility of SimOpt in algorithm development.

ACKNOWLEDGMENTS

This work was partially supported by National Science Foundation Grants DMS-2230023 and OAC-2410950.

REFERENCES

Agrawal, A., R. Verschueren, S. Diamond, and S. Boyd. 2018. "A Rewriting System for Convex Optimization Problems". *Journal of Control and Decision* 5(1):42–60.

Boonsiriphatthanajaroen, N., R. He, L. Liu, T. Ye, and S. G. Henderson. 2024. "Evaluating Solvers for Linearly Constrained Simulation Optimization". In *Proceedings of the 2024 Winter Simulation Conference*, edited by H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. Hunter, and M. D. Rossetti, 3482–3493. Piscataway, New Jersey: IEEE https://doi.org/10.1109/WSC63780.2024.10838622.

Combettes, C. W., and S. Pokutta. 2020, 13–18 Jul. "Boosting Frank-Wolfe by Chasing Gradients". In *Proceedings of the 37th International Conference on Machine Learning*, edited by H. D. III and A. Singh, Volume 119 of *Proceedings of Machine Learning Research*, 2111–2121. Virtual.

Diamond, S., and S. Boyd. 2016. "CVXPY: A Python-embedded Modeling Language for Convex Optimization". *Journal of Machine Learning Research* 17(83):1–5.

Eckman, D. J., S. G. Henderson, and S. Shashaani. 2023. "Diagnostic Tools for Evaluating and Comparing Simulation-Optimization Algorithms". *INFORMS Journal on Computing* 35(2):350–367.

Frank, M., and P. Wolfe. 1956. "An Algorithm for Quadratic Programming". *Naval Research Logistics Quarterly* 3(1-2):95–110.

Frazier, P. I., J. M. Cashore, N. Duan, S. G. Henderson, A. Janmohamed, B. Liu, *et al.* 2022. "Modeling for COVID-19 College Reopening Decisions: Cornell, a Case Study". *Proceedings of the National Academy of Sciences* 119(2):e2112532119 https://doi.org/10.1073/pnas.2112532119.

Iusem, A. N. 2003. "On the Convergence Properties of the Projected Gradient Method for Convex Optimization". *Computational & Applied Mathematics* 22:37–52.

Julien, S. L., and M. Jaggi. 2015, 7–12 Dec. "On the Global Linear Convergence of Frank-Wolfe Optimization Variants". In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, 496–504. Montreal, Canada.

Locatello, F., M. Tschannen, G. Rätsch, and M. Jaggi. 2017. "Greedy Algorithms for Cone Constrained Optimization with Convergence Guarantees". *Advances in Neural Information Processing Systems* 30:773–784.

SimOpt Library 2025. "Simulation Optimization (SimOpt) Library". https://github.com/simopt-admin/simopt/tree/python_dev_boom. Accessed Aug 19, 2025.

Wang, H., H. Lu, and R. Mazumder. 2022. "Frank–Wolfe Methods with an Unbounded Feasible Region and Applications to Structured Learning". *SIAM Journal on Optimization* 32(4):2938–2968 https://doi.org/10.1137/20M1387869.

Wolfe, P. 1970. "Convergence Theory in Nonlinear Programming". In *Integer and Nonlinear Programming*, edited by J. Abadie, 1–36. North-Holland, Amsterdam.

AUTHOR BIOGRAPHIES

NATTHAWUT BOONSIRIPHATTHANAJAROEN is a Ph.D. student in Operations Research and Information Engineering at Cornell University. His research interests are stochastic optimization algorithms and applications of simulation. His email address is nb463@cornell.edu and his homepage is https://natthab.github.io/.

SHANE G. HENDERSON holds the Charles W. Lake, Jr. Chair in Productivity in the School of Operations Research and Information Engineering at Cornell University. His research interests include simulation theory and a range of applications including emergency services. He is an INFORMS Fellow. He is a co-creator of SimOpt, a testbed of simulation optimization problems and solvers. His email address is sgh9@cornell.edu and his homepage is http://people.orie.cornell.edu/shane.