First-order convex (mixed-integer) optimization: FrankWolfe.jl and Boscia.jl

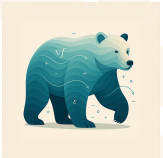
Deborah Hendrych, Mathieu Besançon, Alejandro Carderera, Sébastien Designolle, Jannis Halbey, Daniel Viladrich Herrmannsdoerfer, Dominik Kuzinowicz, Sebastian Pokutta, Hannah Troppens, Elias Wirth, Wenjie Xiao

hendrych@zib.de

JuMP-dev 2026 · June 01, 2026

Berlin Mathematics Research Center

MATH+



Outline

Frank-Wolfe

Frank and Wolfe 1956; Levitin and Polyak 1966; Braun et al. 2022

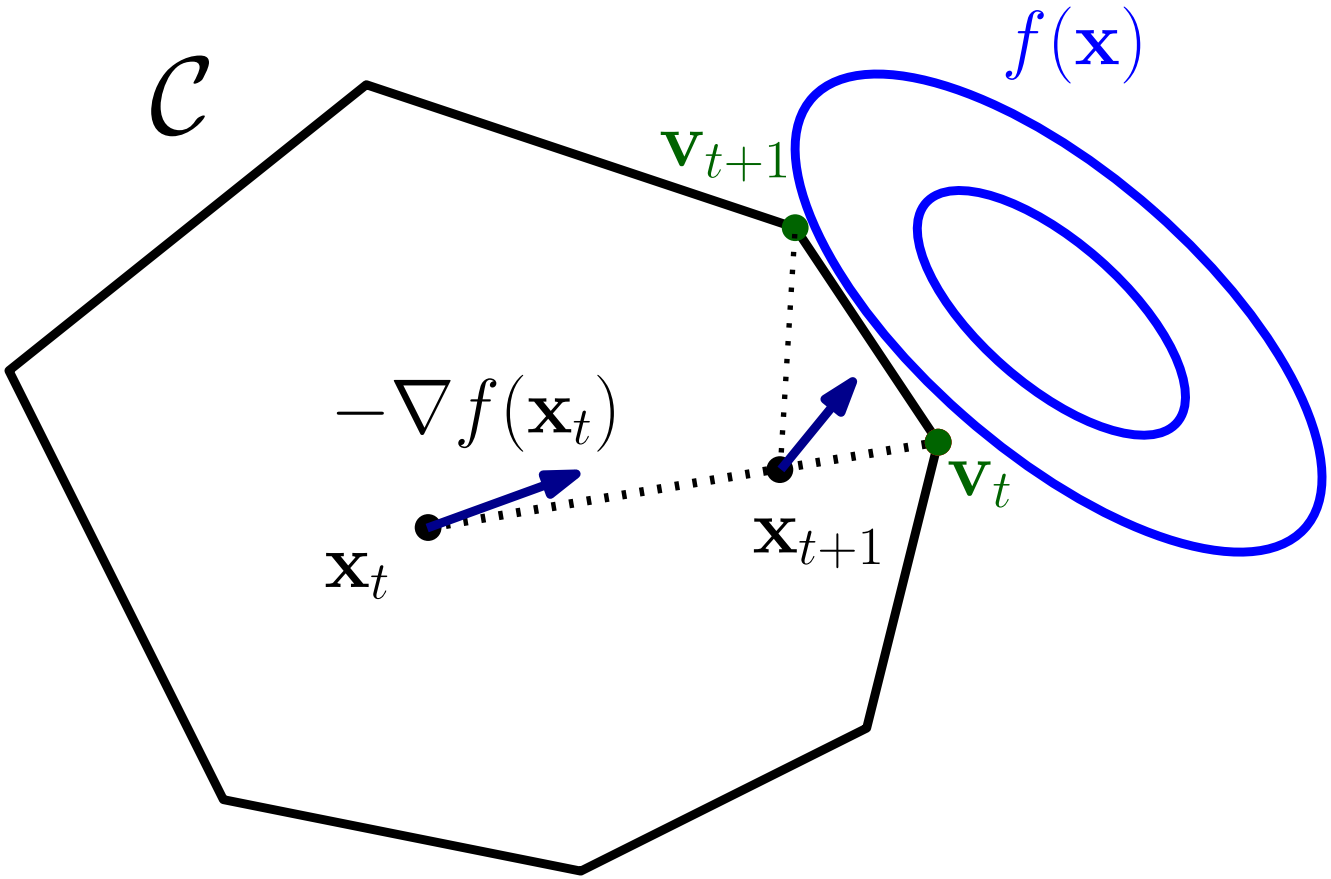
$$\min f(\mathbf{x})$$
$$\text{s.t. } \mathbf{x} \in \mathcal{C}$$

where

- f is a convex, (L-smooth) function.
- Oracle access to f and ∇f.
- $\mathcal{C} \subset \mathbb{R}^n$ is a compact convex set.
- $\mathcal{C}$ admits an *efficient* Linear Minimization Oracle (LMO):

$$\mathbf{v} = \arg\min_{\mathbf{y} \in \mathcal{C}} \langle \mathbf{d}, \mathbf{y} \rangle$$

Frank-Wolfe



$\mathbf{x}_t$

- $\mathbf{x}_t$ is always feasible.
- iterate $\mathbf{x}_t$ can be written as a convex combination of the extreme points of C, often stored as *active set*.
- Let $\mathbf{x}^*$ an optimal solution. Then,

$$f(\mathbf{x}_t) - f(\mathbf{x}^*) \leq \langle \nabla f(\mathbf{x}_t), \mathbf{x}_t - \mathbf{x}^* \rangle \leq \max_{\mathbf{v} \in C} \langle \nabla f(\mathbf{x}_t), \mathbf{x}_t - \mathbf{v} \rangle := g(\mathbf{x}_t)$$

  is called the *Frank-Wolfe gap*.

FrankWolfe.jl package

Mixed-Integer Convex Optimization

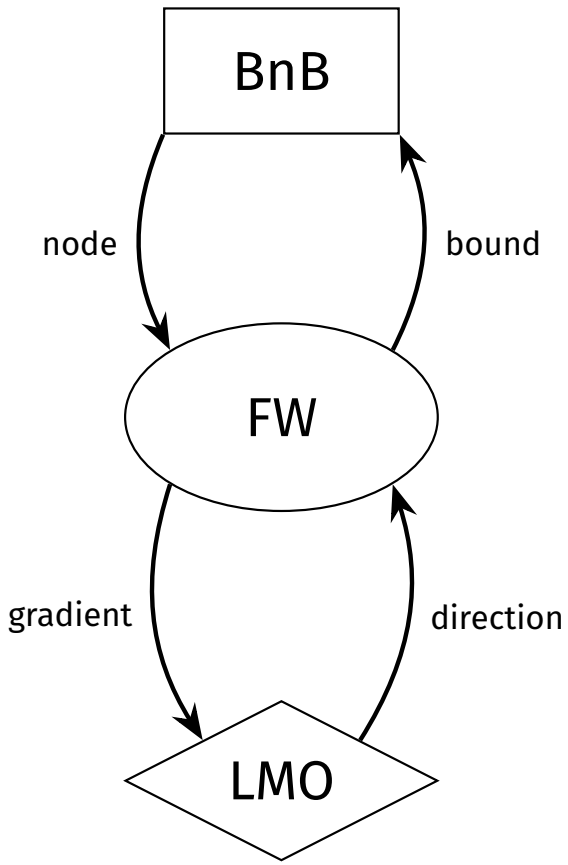
$$\begin{aligned} \min\ & f(\mathbf{x}) \\ \text{s.t. } & \mathbf{x} \in \mathcal{P} \\ & x_j \in \mathbb{Z} \quad \forall j \in J \end{aligned}$$

where

- f is a convex, (L-smooth) function.
- $\mathcal{P} \subset \mathbb{R}^n$ is compact, convex, (polyhedral) set.
- $J \subset [n]$ is a set of indices.

BnB with Frank-Wolfe: Boscia

Hendrych et al. 2025



$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{P} \\ & x_j \in \mathbb{Z} \quad \forall j \in J \end{aligned}$$

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in \text{conv}(\mathcal{P} \cap \mathbb{Z}_J) \end{aligned}$$

$$\begin{aligned} \min_{\mathbf{v}} \quad & \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle \\ \text{s.t.} \quad & \mathbf{v} \in \mathcal{P} \cap \mathbb{Z}_J \end{aligned}$$

Branch-and-Bound with Frank-Wolfe: `Boscia.jl`

Key Features:

- Precision-adaptive Frank-Wolfe allows for a trade-off between accuracy and computational cost.
- Warm-starting nodes via splitting the active set of the parent node.
- Aggressive reuse of information: Lazification & shadow set techniques
- Built-in heuristic: Integer feasible solutions from root node.
- No closed form of the objective function required.
- No comprehensive description of the feasible region required.

FrankWolfe.jl

```
f(x) = norm(x - xp)^2
function grad!(storage, x)
    @. storage = 2 * (x - xp)
    return nothing
end
```

FrankWolfe.jl

```
f(x) = norm(x - xp)^2
function grad!(storage, x)
    @. storage = 2 * (x - xp)
    return nothing
end

model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x[1:n])
@constraint(model, x[1:n] ≥ 0)
@constraint(model, sum(x) == lmo_radius)
lmo_jump = FrankWolfe.MathOptLMO(model.moi_backend)
```

FrankWolfe.jl

```
f(x) = norm(x - xp)^2
function grad!(storage, x)
    @. storage = 2 * (x - xp)
    return nothing
end

model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x[1:n])
@constraint(model, x[1:n] ≥ 0)
@constraint(model, sum(x) == lmo_radius)
lmo_jump = FrankWolfe.MathOptLMO(model.moi_backend)

lmo_radius = 2.5
lmo = FrankWolfe.ProbabilitySimplexLMO(lmo_radius)
x00 = FrankWolfe.compute_extreme_point(lmo, zeros(n))
gradient = collect(x00)
```

FrankWolfe.jl

```
f(x) = norm(x - xp)^2
function grad!(storage, x)
    @. storage = 2 * (x - xp)
    return nothing
end

model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x[1:n])
@constraint(model, x[1:n] ≥ 0)
@constraint(model, sum(x) == lmo_radius)
lmo_jump = FrankWolfe.MathOptLMO(model.moi_backend)

lmo_radius = 2.5
lmo = FrankWolfe.ProbabilitySimplexLMO(lmo_radius)
x00 = FrankWolfe.compute_extreme_point(lmo, zeros(n))
gradient = collect(x00)

x, v, primal, dual_gap, status, trajectory = FrankWolfe.frank_wolfe(
    f, grad!, lmo, collect(copy(x00)),
    max_iteration=k, line_search=FrankWolfe.Secant(),
    print_iter=k / 10, verbose=true, trajectory=true,
);
```

Boscia.jl

```
model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x[1:n], Int)
@constraint(model, x[1:n] .≥ 0)
@constraint(model, x[1:n] .≤ 1)
lmo_jump = FrankWolfe.MathOptLMO(model.moi_backend)
```

Boscia.jl

```
model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x[1:n], Int)
@constraint(model, x[1:n] .≥ 0)
@constraint(model, x[1:n] .≤ 1)
lmo_jump = FrankWolfe.MathOptLMO(model.moi_backend)

int_vars = collect(1:n)
lbs = zeros(n)
ubs = ones(n)
sblmo = FrankWolfe.ZeroOneHypercubeLMO()
lmo = Boscia.ManagedBoundedLMO(sblmo, lbs[int_vars], ubs[int_vars], int_vars, n)
```

Boscia.jl

```
model = Model(HiGHS.Optimizer)
set_silent(model)
@variable(model, x[1:n], Int)
@constraint(model, x[1:n] .≥ 0)
@constraint(model, x[1:n] .≤ 1)
lmo_jump = FrankWolfe.MathOptLMO(model.moi_backend)

int_vars = collect(1:n)
lbs = zeros(n)
ubs = ones(n)
sblmo = FrankWolfe.ZeroOneHypercubeLMO()
lmo = Boscia.ManagedBoundedLMO(sblmo, lbs[int_vars], ubs[int_vars], int_vars, n)

settings = Boscia.create_default_settings()
settings.branch_and_bound[:verbose] = true
x, _, result = Boscia.solve(f, grad!, lmo, settings=settings);
```

Outlook

Practicality and UI

- Connect to `MathOptInterface` (MOI) and `JuMP`.
- Web service.
- Precompiled and executable version.

Outlook

Practicality and UI

- Connect to `MathOptInterface` (MOI) and `JuMP`.
- Web service.
- Precompiled and executable version.

Capabilities

- Preprocessing and propagation.
- Non-convex objective function.

Thank you for your attention!

FrankWolfe.jl

Boscia.jl

References I

Ahipaşaoğlu, Selin Damla (2021). “A branch-and-bound algorithm for the exact optimal experimental design problem”. In: *Statistics and Computing* 31.5, p. 65.

Braun, Gábor, Alejandro Carderera, Cyrille W Combettes, Hamed Hassani, Amin Karbasi, Aryan Mokhtari, and Sebastian Pokutta (2022). “Conditional gradient methods”. In: *arXiv preprint arXiv:2211.14103*.

Buchheim, Christoph, Marianna De Santis, Francesco Rinaldi, and Long Trieu (2018). “A Frank–Wolfe based branch-and-bound algorithm for mean-risk optimization”. In: *Journal of Global Optimization* 70.3, pp. 625–644.

Frank, Marguerite and Philip Wolfe (1956). “An algorithm for quadratic programming”. In: *Naval research logistics quarterly* 3.1-2, pp. 95–110.

Hendrych, Deborah, Hannah Troppens, Mathieu Besançon, and Sebastian Pokutta (June 28, 2025). “Convex Integer Optimization with Frank-Wolfe Methods”. In: *Mathematical Programming Computation*. DOI: 10.1007/s12532-025-00288-w. arXiv: 2208.11010 [math.OC]. URL: https://link.springer.com/article/10.1007/s12532-025-00288-w.

Levitin, Evgeny S and Boris T Polyak (1966). “Constrained minimization methods”. In: *USSR Computational mathematics and mathematical physics* 6.5, pp. 1–50.

Li, Yongchun, Marcia Fampa, Jon Lee, Feng Qiu, Weijun Xie, and Rui Yao (2024). “D-Optimal Data Fusion:: Exact and Approximation Algorithms”. In: *INFORMS Journal on Computing* 36.1, pp. 97–120.

References II

Mexi, Gioni, Deborah Hendrych, Sébastien Designolle, Mathieu Besançon, and Sebastian Pokutta (Aug. 2, 2025). *A Frank-Wolfe-based Primal Heuristic for Quadratic Mixed-integer Optimization*. arXiv: 2508.01299 [math.OC].

Ponte, Gabriel, Marcia Fampa, and Jon Lee (2025). "Branch-and-bound for D-Optimality with fast local search and variable-bound tightening". In: *Mathematical Programming*, pp. 1–38.

Pukelsheim, Friedrich (2006). *Optimal design of experiments*. SIAM.