Smooth Constraint Convex Minimization via Conditional Gradients

Sebastian Pokutta

H. Milton Stewart School of Industrial and Systems Engineering
Center for Machine Learning @ GT (ML@GT)
Algorithm and Randomness Center (ARC)
Georgia Institute of Technology

Tokyo, 03/2019

Georgia Tech Machine Learning

twitter: @spokutta

Joint Work with… (in random order)

Alexandre D'Aspremont

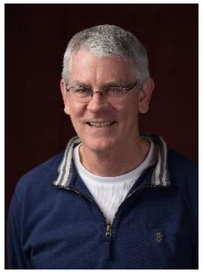
Thomas Kerdreux

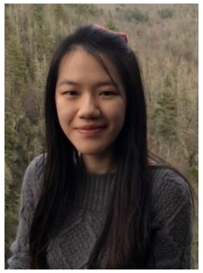
Gábor Braun

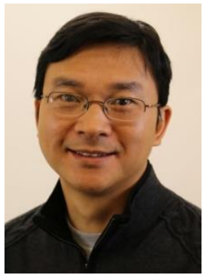
Cyrille Combettes

Swati Gupta

Robert Hildebrand


Stephen Wright


Yi Zhou


George Lan

Dan Tu

Daniel Zink

Georgia Tech Machine Learning

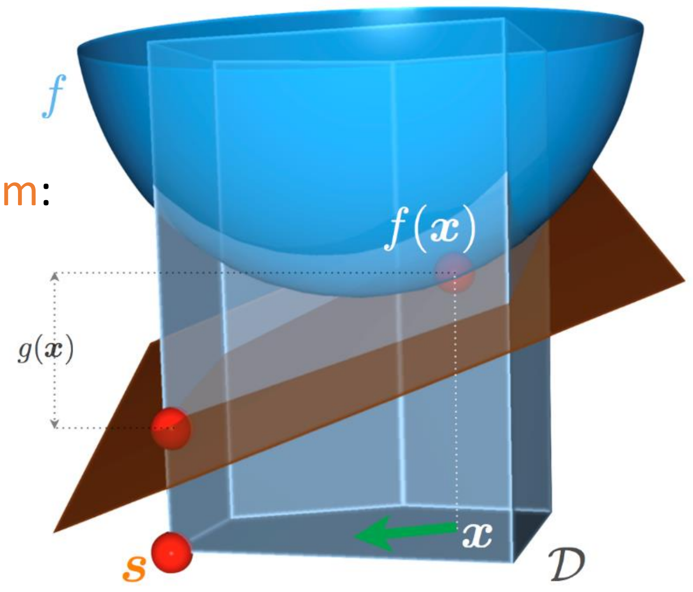
(Constraint) Convex Optimization

Convex Optimization:

Given a feasible region P solve the optimization problem:

$$\mathrm{Min}_{x \in P}\ f(x),$$

where f is a convex function (+ extra properties).



Source: [Jaggi 2013]

Our setup.

1. Access to P. Linear Optimization (LO) oracle: Given linear objective *c*

$$x \leftarrow \mathrm{argmin}_{v \in P} c^T v$$

2. Access to f. First-Order (FO) oracle: Given x return

$$\nabla f(x) \text{ and } f(x)$$

=> Complexity of convex optimization relative to LO/FO oracle

Georgia Tech Machine Learning

Why would you care for constraint convex optimization?

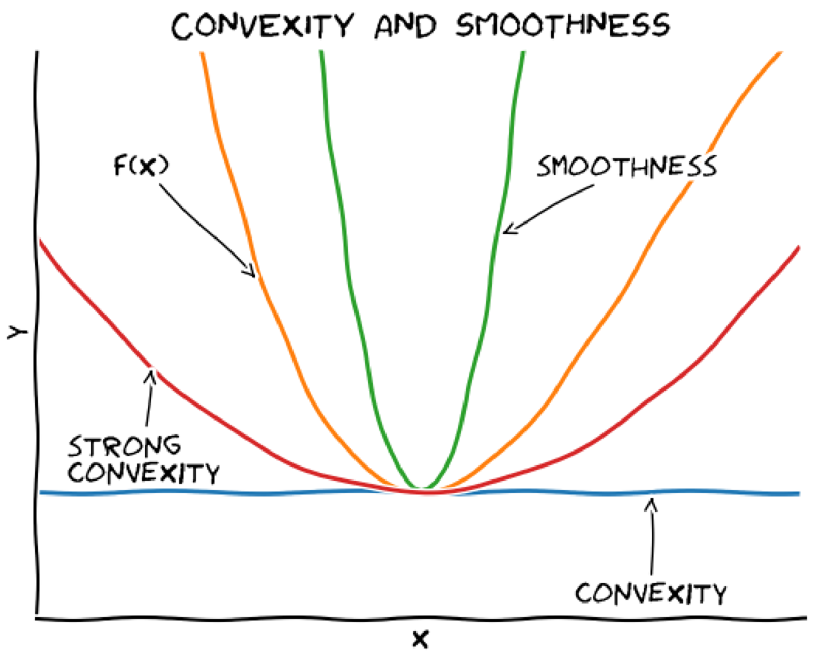
Setup captures various problems in Machine Learning, e.g.:

1. OCR (Structured SVM Training)
    1. Marginal polytope over chain graph of letters of word and quadratic loss
2. Video Co-Localization
    1. Flow polytope and quadratic loss
3. Lasso
    1. Scaled ℓ_1-ball and quadratic loss (regression)
4. Regression over structured objects
    1. Regression over convex hull of combinatorial atoms
5. Approximation of distributions
    1. Bayesian inference, sequential kernel herding, …

Smooth Convex Optimization 101

Georgia Tech Machine Learning

Basic notions



Let $f: R^n \rightarrow R$ be a function. We will use the following basic concepts:

Smoothness. $f(y) \leq f(x) + \nabla f(x)^T (y-x) + \frac{L}{2}||x-y||^2$

Convexity. $f(y) \geq f(x) + \nabla f(x)^T (y-x)$

Strong Convexity. $f(y) \geq f(x) + \nabla f(x)^T (y-x) + \frac{\mu}{2}||x-y||^2$

=> Use for optimization unclear. Next step: Operationalize notions!

Measures of Progress: Smoothness and Idealized Gradient Descent

Consider an iterative algorithm of the form:

$$x_{t+1} \leftarrow x_t - \eta_t d_t$$

By definition of smoothness. $f(x_t) - f(x_{t+1}) \geq \eta_t \nabla f(x_t)^T d_t - \eta_t^2 \frac{L}{2} \left|\left|d_t\right|\right|^2$

Smoothness induces primal progress. Optimizing right-hand side:

$$f(x_t) - f(x_{t+1}) \geq \frac{(\nabla f(x_t)^T d_t)^2}{2L\left|\left|d_t\right|\right|^2} \qquad \text{for} \qquad {\eta_t}^* = \frac{(\nabla f(x_t)^T d_t)}{L\left|\left|d_t\right|\right|^2}$$

Idealized Gradient Descent (IGD). Choose $d_t \leftarrow x_t - x^*$ **(non-deterministic!)**

$$f(x_t) - f(x_{t+1}) \geq \frac{(\nabla f(x_t)^T (x_t - x^*))^2}{2L\left|\left|x_t - x^*\right|\right|^2} \qquad \text{for} \qquad {\eta_t}^* = \frac{(\nabla f(x_t)^T (x_t - x^*))}{L\left|\left|x_t - x^*\right|\right|}$$

Georgia Tech Machine Learning

Measures of Optimality: Convexity

Recall convexity: $f(y) \geq f(x) + \nabla f(x)^T (y - x)$

Primal bound from Convexity. $x \leftarrow x_t$ and $y \leftarrow x^* \in \operatorname{argmin}_{x \in P} f(x)$:

$$h_t := f(x_t) - f(x^*) \leq \nabla f(x_t)^T (x_t - x^*)$$

Plugging this into the progress from IGD and $||x_t - x^*|| \leq ||x_0 - x^*||$.

$$f(x_t) - f(x_{t+1}) \geq \frac{\left(\nabla f(x_t)^T (x_t - x^*)\right)^2}{2L||x_t - x^*||^2} \geq \frac{h_t^2}{2L||x_0 - x^*||^2}$$

Rearranging provides contraction and convergence rate.

$$h_{t+1} \leq h_t \cdot \left(1 - \frac{h_t}{2L||x_0 - x^*||^2}\right) \quad \Rightarrow \quad h_T \leq \frac{2L||x_0 - x^*||^2}{T + 4}$$

Measures of Optimality: Strong Convexity

Recall strong convexity: $f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \left|\left|x - y\right|\right|^2$

Primal bound from Strong Convexity. $x \leftarrow x_t$ and $y \leftarrow x_t - \gamma(x_t - x^*)$

$$h_t := f(x_t) - f(x^*) \leq \frac{\left(\nabla f(x_t)^T (x_t - x^*)\right)^2}{2\mu \left|\left|x_t - x^*\right|\right|^2}$$

Plugging this into the progress from IGD.

$$f(x_t) - f(x_{t+1}) \geq \frac{\left(\nabla f(x_t)^T (x_t - x^*)\right)^2}{2L \left|\left|x_t - x^*\right|\right|^2} \geq \frac{\mu}{L} h_t$$

Rearranging provides contraction and convergence rate.

$$h_{t+1} \leq h_t \cdot \left(1 - \frac{\mu}{L}\right) \quad \Rightarrow \quad h_T \leq e^{-\frac{\mu}{L}T} \cdot h_0$$

From IGD to actual algorithms

Consider an algorithm of the form:

$$x_{t+1} \leftarrow x_t - \eta_t d_t$$

Scaling condition (Scaling). Show there exist α_t with

$$\frac{\nabla f(x_t)^T d_t}{||d_t||} \geq \alpha_t \frac{\nabla f(x_t)^T (x_t - x^*)}{||x_t - x^*||}$$

=> Lose an α_t^2 factor in iteration t. Bounds and rates follow immediately.

Example. (Vanilla) Gradient Descent with $d_t \leftarrow \nabla f(x_t)$

$$\frac{\nabla f(x_t)^T d_t}{||d_t||} = ||\nabla f(x_t)||^2 \geq 1 \cdot \frac{\nabla f(x_t)^T (x_t - x^*)}{||x_t - x^*||}$$

=> TODAY: No more convergences proofs. Just establishing (Scaling).

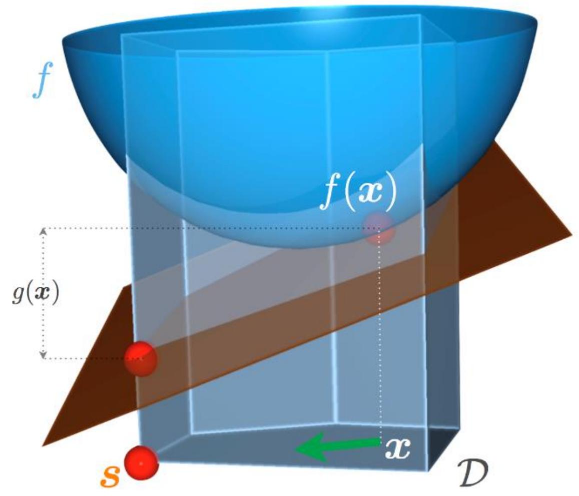
Conditional Gradients
(a.k.a. Frank-Wolfe Algorithm)

Conditional Gradients a.k.a. Frank-Wolfe Algorithm

Algorithm 1 Frank-Wolfe Algorithm [Frank and Wolfe, 1956]

Input: smooth convex f function with curvature C, $x_1 \in P$
Output: x_t points in P

```
1: for $t = 1$ to $T - 1$ do
2:     $v_t \leftarrow \mathrm{LP}_P(\nabla f(x_t))$
3:     $x_{t+1} \leftarrow (1 - \gamma_t)x_t + \gamma_t v_t$ with $\gamma_t := \frac{2}{t+2}$
4: end for
```

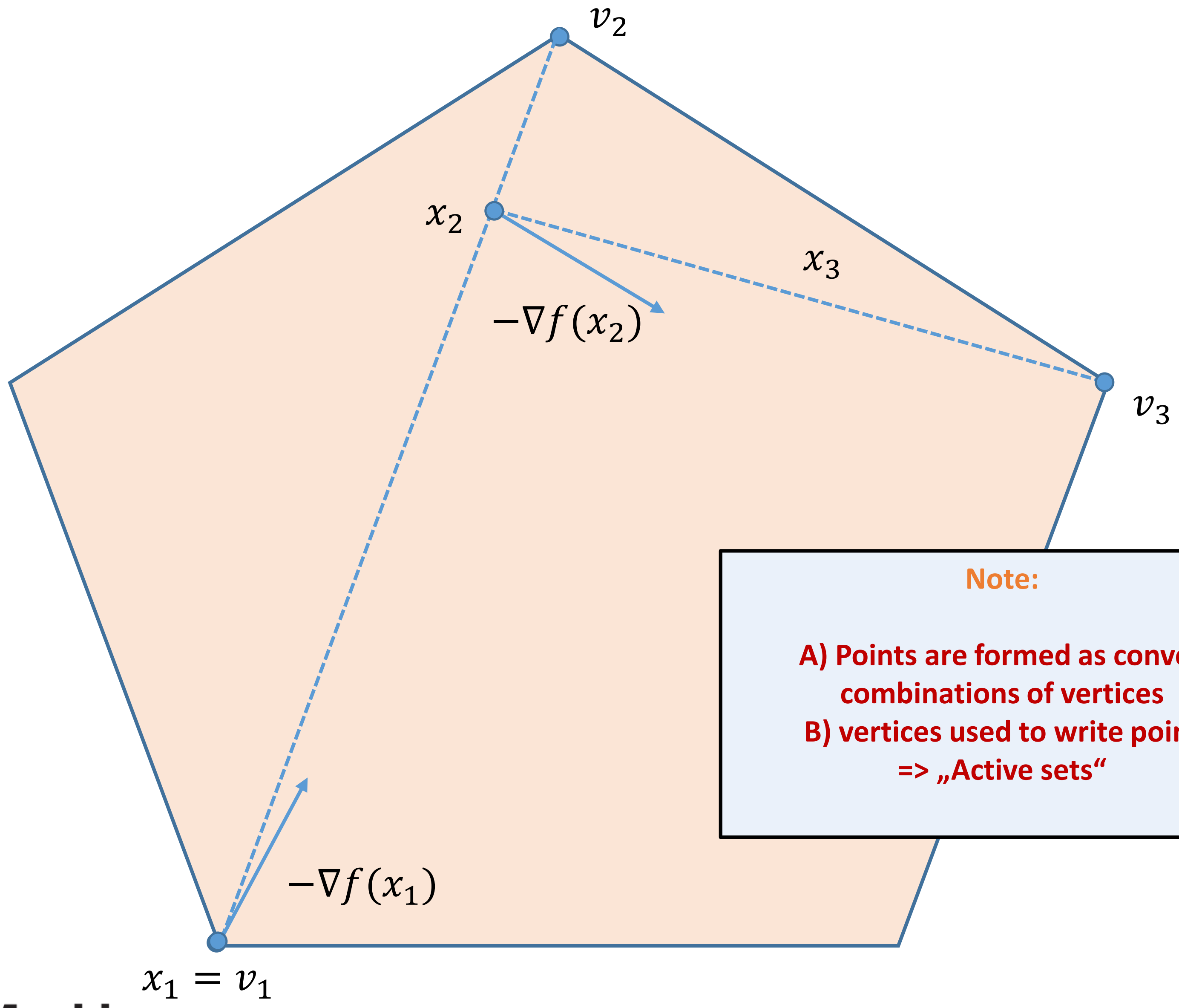


Source: [Jaggi 2013]

1. Advantages
   1. Extremely simple and robust: no complicated data structures to maintain
   2. Easy to implement: requires only a linear optimization oracle (first order method)
   3. Projection-free: feasibility via linear optimization oracle
   4. Sparse distributions over vertices: optimal solution is convex comb. (enables sampling)
2. Disadvantages
   1. Suboptimal convergence rate of $O(\frac{1}{T})$ in the worst-case

=> Despite suboptimal rate often used because of simplicity

Conditional Gradients a.k.a. Frank-Wolfe Algorithm

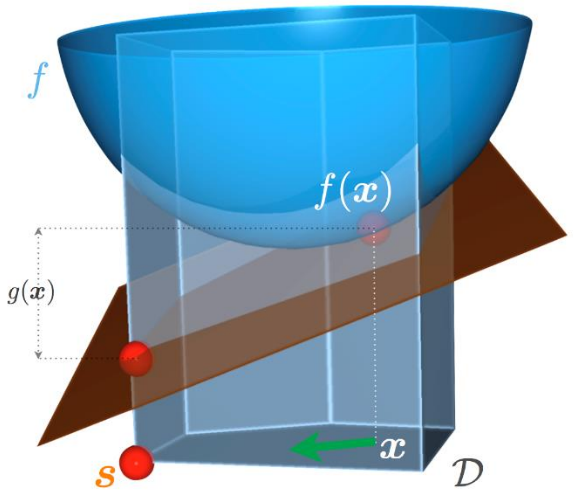


Note:

A) Points are formed as convex combinations of vertices
B) vertices used to write point => „Active sets“

Conditional Gradients a.k.a. Frank-Wolfe Algorithm

```
Algorithm 1 Frank-Wolfe Algorithm [Frank and Wolfe, 1956]
Input: smooth convex $f$ function with curvature C, $x_1 \in P$
Output: $x_t$ points in $P$
1: for $t = 1$ to $T - 1$ do
2:     $v_t \leftarrow \mathrm{LP}_P(\nabla f(x_t))$
3:     $x_{t+1} \leftarrow (1 - \gamma_t)x_t + \gamma_t v_t$ with $\gamma_t := \frac{2}{t+2}$
4: end for
```



Source: [Jaggi 2013]

Establishing (Scaling).

FW algorithm takes direction $d_t = x_t - v_t$. Observe

$$\nabla f(x)^T(x_t - v_t) \geq \nabla f(x)^T(x_t - x^*)$$

Hence with $\alpha_t = \frac{||x_t - x^*||}{D}$ with *D* diameter of *P*:

$$\frac{\nabla f(x)^T(x_t - v_t)}{||x_t - v_t||} \geq \frac{||x_t - x^*||}{D} \cdot \frac{\nabla f(x)^T(x_t - x^*)}{||x_t - x^*||}$$

=> This α_t is sufficient for $O(\frac{1}{t})$ convergence but better??

The strongly convex case
Linear convergence in special cases

If f is strongly convex we would expect a linear rate of convergence.

Obstacle.

$$\frac{\nabla f(x)^T(x_t - v_t)}{||x_t - v_t||} \geq \frac{\boldsymbol{||x_t - x^*||}}{D} \cdot \frac{\nabla f(x)^T(x_t - x^*)}{||x_t - x^*||}$$

Special case $x^* \in \mathrm{rel.int}(P)$, say $B(x^*, 2r) \subseteq P$. Then:

Theorem [Marcotte, Guélat '86]. After a few iterations

$$\frac{\nabla f(x)^T(x_t - v_t)}{||x_t - v_t||} \geq \frac{r}{D} \cdot \frac{\nabla f(x)^T(x_t - x^*)}{||x_t - x^*||}$$

and linear convergence follows via (Scaling).

The strongly convex case
Is linear convergence in general possible?

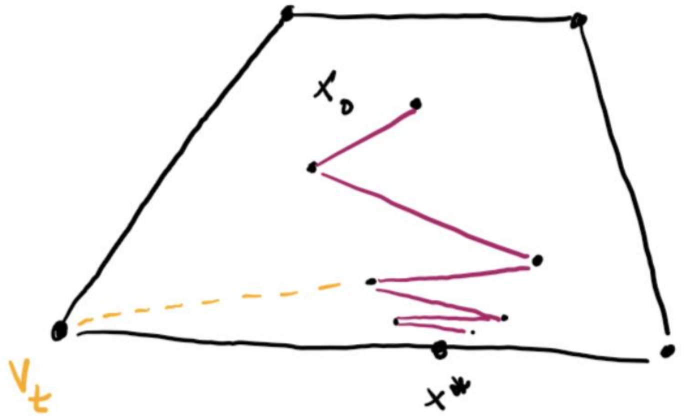
(Vanilla) Frank-Wolfe cannot achieve linear convergence in general:

Theorem [Wolfe '70]. x^* on boundary of *P*. For any $\delta > 0$ for infinitely many *t*:

$$f(x_t) - f(x^*) \geq \frac{1}{t^{1+\delta}}$$

Issue: zig-zagging (b/c first order opt)

[Wolfe '70] proposed Away Steps



The strongly convex case
Linear convergence in general

First linear convergence result (in general) due to [Garber, Hazan '13]

1. Simulating (theoretically efficiently) a stronger oracle rather using Away Steps
2. Involved constants are *extremely large* => algorithm unimplementable

Linear convergence for implementable variants due to [Lacoste-Julien, Jaggi '15]

1. (Dominating) Away-steps are enough
2. Includes most known variants: Away-Step FW, Pairwise CG, Fully-Corrective FW, Wolfe's algorithm, …
3. Key ingredient: There exists $w(P)$ (depending on polytope P (only!)) s.t.

$$\nabla f(x)^T(a_t - v_t) \geq w(P) \frac{\nabla f(x)^T(x_t - x^*)}{||x_t - x^*||}$$

($d_t = a_t - v_t$ is basically the direction that either variant dominates)

=> Linear convergence via (Scaling)

Many more variants and results…

Recently there has been a lot of work on Conditional Gradients, e.g.,

1. Linear convergence for conditional gradient sliding [Lan, Zhou '14]
2. Linear convergence for (some) non-strongly convex functions [Beck, Shtern '17]
3. Online FW [Hazan, Kale '12, Chen et al '18]
4. Stochastic FW [Reddi et al '16] and Variance-Reduced Stochastic FW [Hazan, Luo '16, Chen et al '18]
5. In-face directions [Freund, Grigas '15]
6. Improved convergence under sharpness [Kerdreux, D'Aspremont, P. '18]

… and *many more!!*

=> Very competitive and versatile in real-world applications

Revisiting Conditional Gradients

Bottleneck 1: Cost of Linear Optimization
Drawbacks in the context of hard feasible regions

Basic assumption of conditional gradient methods:

Linear Optimization is cheap

As such accounted for as $O(1)$. This assumption is not warranted if:

1. Linear Program of feasible region is huge
    1. Large shortest path problems
    2. Large scheduling problems
    3. Large-scale learning problems
2. Optimization over feasible region is NP-hard
    1. TSP tours
    2. Packing problems
    3. Virtually *every* real-world combinatorial optimization problem

Rethinking CG in the context of expensive oracle calls

Basic assumption for us:

*Linear Optimization is **not** cheap*

(Think: hard IP can easily require an hour to be solved => one call/it unrealistic)

1. Questions:
    1. Is it necessary to call the oracle in each iteration?
    2. Is it necessary to compute *(approximately) optimal* solutions?
    3. Can we reuse information?

2. Theoretical requirements
    1. Achieve identical convergence rates, otherwise any speedup will be washed out

3. Practical requirements
    1. Make as few oracle calls as possible

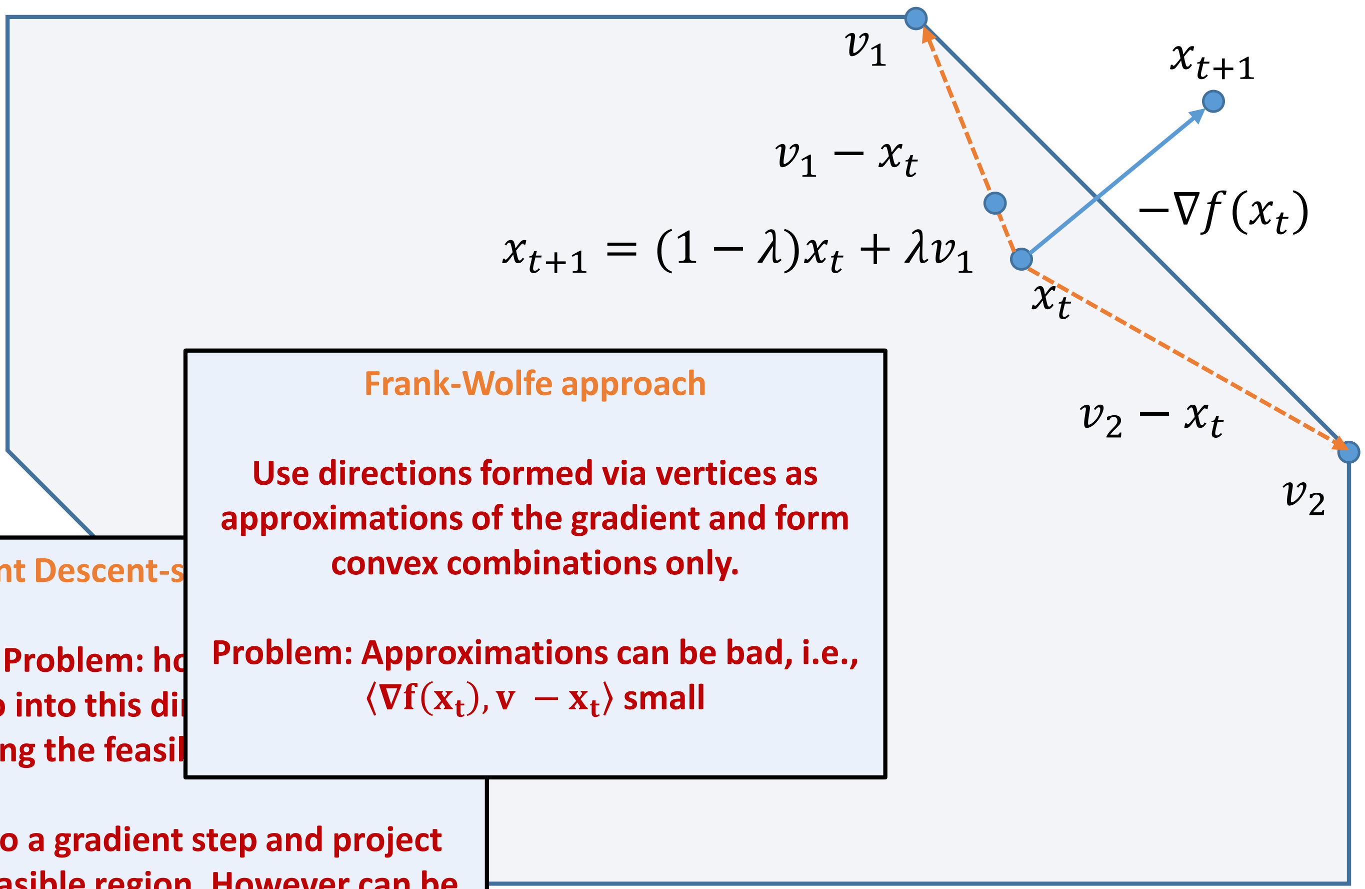
Lazification approach of [BPZ 2017] using weaker oracle

Oracle 1 Weak Separation Oracle $\mathrm{LPsep}_P(c, x, \Phi, K)$

Require: $c \in \mathbb{R}^n$ linear objective, $x \in P$ point, $K \geq 1$ accuracy, $\Phi > 0$ objective value;
Ensure: Either (1) $y \in P$ vertex with $c(x - y) > \Phi/K$, or (2) **false**: $c(x - z) \leq \Phi$ for all $z \in P$.

1. Interpretation of Weak Separation Oracle: *Discrete Gradient Directions*
    1. Either a new point $y \in P$ that improves the current objective by at least $\frac{\Phi}{K}$ (positive call)
    2. Or it asserts that all other points $z \in P$ improve no more than Φ (negative call)
2. Lazification approach of [Braun, P., Zink '17]
    1. Use weaker oracle that allows for caching and early termination (no more expensive than LP)
    2. Advantage: huge speedups in wall-clock time when LP is hard to solve
        1. For hard LPs speedups can be as large as 10^7
    3. Disadvantage: weak separation oracle produces even weaker approximations than LP oracle
        1. Actual progress in iterations can be worse than with LP oracle
        2. Advantage vanishes if LP is very cheap and can be actually worse than original algorithm
        3. Caching is not "smart": it simple iterates over the already seen vertices
3. Optimal complexity for Weak Separation Oracle [Braun, Lan, P., Zhou '17]

Bottleneck 2: Quality of gradient approximation Frank-Wolfe vs. Projected Gradient Descent



Frank-Wolfe approach

Use directions formed via vertices as approximations of the gradient and form convex combinations only.

Problem: Approximations can be bad, i.e., $\langle \nabla f(x_t), v - x_t \rangle$ small

Gradient Descent-s

Problem: ho
can we go into this di
leaving the feasil

Solution: do a gradient step and project back into feasible region. However can be *very* expensive

$\Rightarrow$ **Tradeoff between ensured feasibility and quality of gradient approximations!**

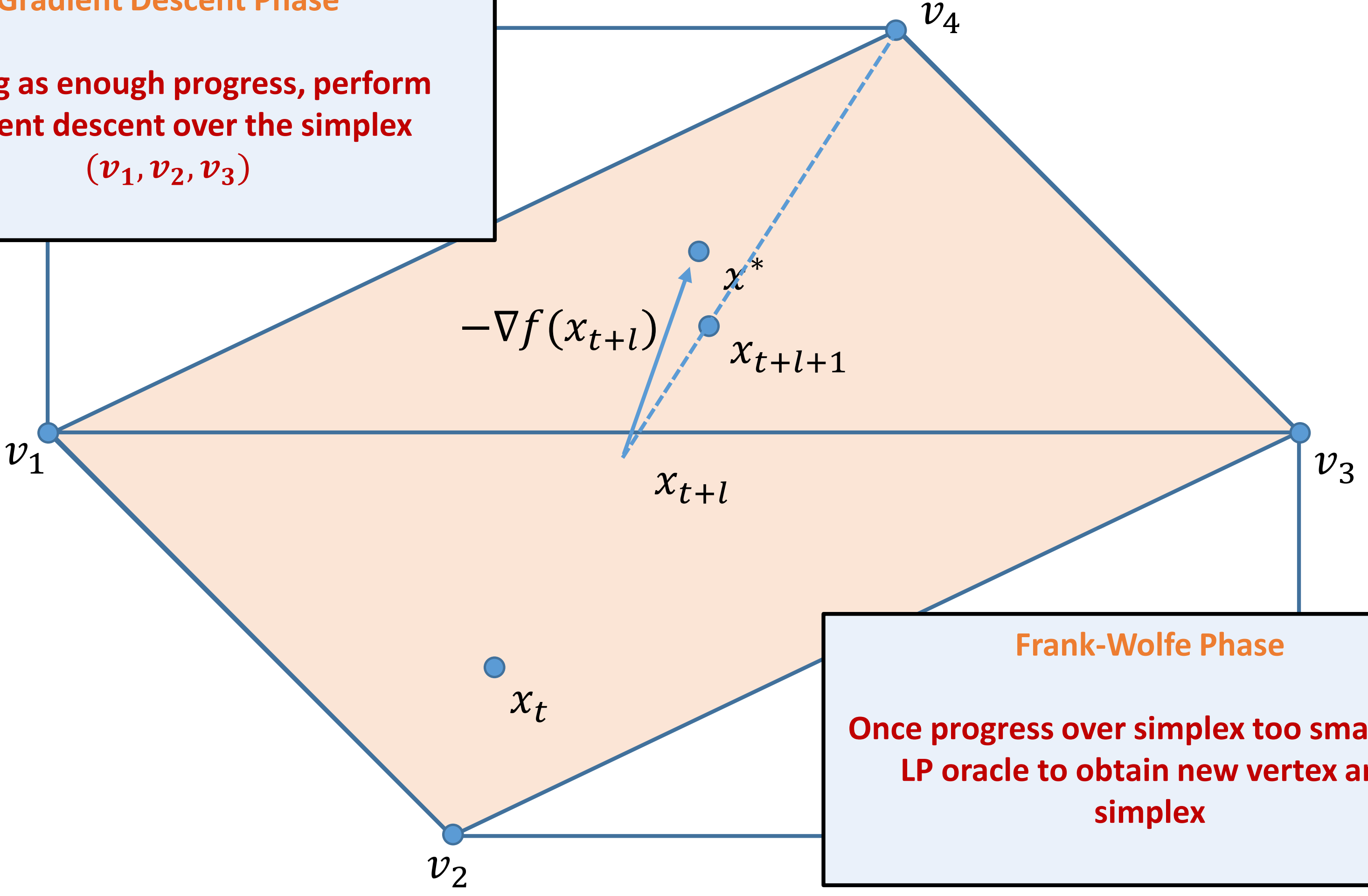
Georgia Tech Machine Learning

Bringing Conditional Gradients as close as possible to Gradient Descent

Blending of gradient steps and Frank-Wolfe steps

Gradient Descent Phase

As long as enough progress, perform gradient descent over the simplex (v_1, v_2, v_3)



Frank-Wolfe Phase

Once progress over simplex too small, call LP oracle to obtain new vertex and simplex

Staying Projection-free: The Simplex Descent Oracle (SiDO)

```
**Oracle 1** Simplex Descent Oracle SiDO$(x, S, f)$
**Input:** finite set $S \subseteq \mathbb{R}^n$, point $x \in \operatorname{conv} S$, smooth convex function $f\colon \operatorname{conv} S \to \mathbb{R}^n$
**Output:** finite set $S' \subseteq S$, point $x' \in \operatorname{conv} S'$ satisfying either (1) $f(x') \leq f(x)$ and $S' \neq S$, or
    (2) $f(x) - f(x') \geq [\max_{u,v\in S} \nabla f(x)(u-v)]^2/(4L_{f_S})$.
```

1. Interpretation of Simplex Descent Oracle: *Single progress step on simplex*
    1. Either reduce size of set S by at least 1 if function is not increasing (guaranteed progress in size)
    2. Or make a descent step with a guaranteed improvement relative to best approximate direction (guaranteed progress in function value)

2. Various implementations of SiDO
    1. Most basic via projected gradient descent (PGD) however requires projections over a low-dimensional simplices
    2. Better version via **new(!!)** projection-free algorithm over simplex *(Simplex Gradient Descent)* with cost linear in the size of the active set per iteration

The algorithm: Blended Conditional Gradients

Algorithm 1 Blended Conditional Gradients (BCG)

```
Input: smooth convex function $f$, start vertex $x_0 \in P$, weak separation oracle $\mathrm{LPsep}_P$, accuracy $K \geq 1$
Output: points $x_t$ in $P$ for $t = 1, \ldots, T$
 1: $\Phi_0 \leftarrow \max_{v \in P} \nabla f(x_0)(x_0 - v)/2$        {Initial dual gap estimate}
 2: $S_0 \leftarrow \{x_0\}$
 3: for $t = 0$ to $T - 1$ do
 4:     $v_t^A \leftarrow \mathrm{argmax}_{v \in S_t} \nabla f(x_t) v$
 5:     $v_t^{FW-S} \leftarrow \mathrm{argmin}_{v \in S_t} \nabla f(x_t) v$
 6:     if $\nabla f(x_t)(v_t^A - v_t^{FW-S}) \geq \Phi_t$ then
 7:         $x_{t+1}, S_{t+1} \leftarrow \mathrm{SiDO}(x_t, S_t)$
 8:         $\Phi_{t+1} \leftarrow \Phi_t$
20:     end if
21: end for
```

Gradient Descent Phase

As long as enough progress, perform gradient descent over the simplex

Dual Gap Update Phase

If neither SiDO nor Frank-Wolfe steps can progress enough update dual gap

Frank-Wolfe Phase

Once progress too small, call LPSep oracle for new vertex and simplex

Main Theorem

You basically get what you expect:

Theorem. [Braun, P., Tu, Wright '18] Assume f is convex and smooth over the polytope P with curvature C and geometric strong convexity μ. Then Algorithm 1 ensures:

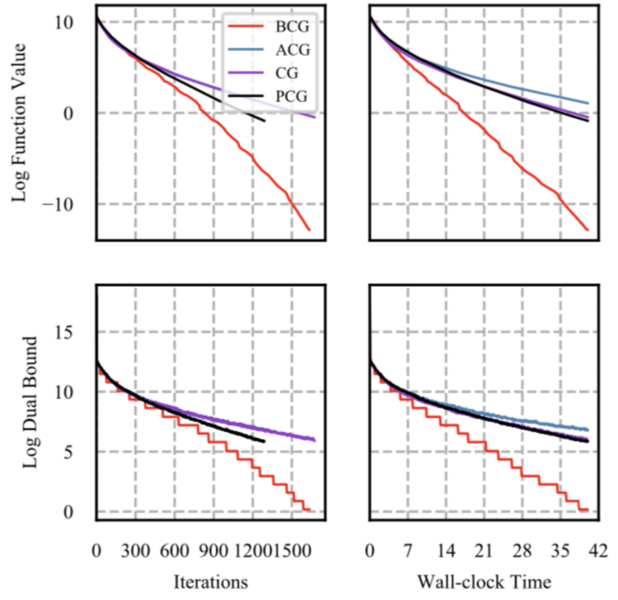
$$f(x_t) - f(x^*) \leq \varepsilon \qquad \text{for } t \geq \Omega\left(\frac{C}{\mu} \log \frac{\Phi_0}{\varepsilon}\right),$$

where x^* is an optimal solution to f over P and $\Phi_0 \geq f(x_0) - f(x^*)$.

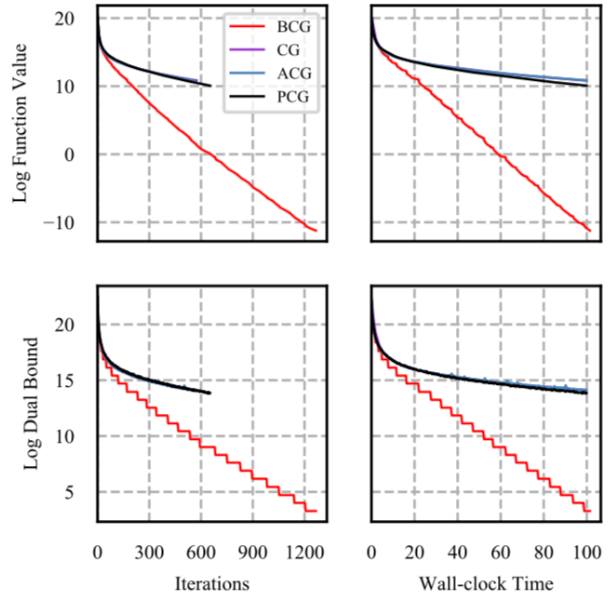
(For previous empirical work with similar idea see also [Rao, Shah, Wright '15])

Computational Results

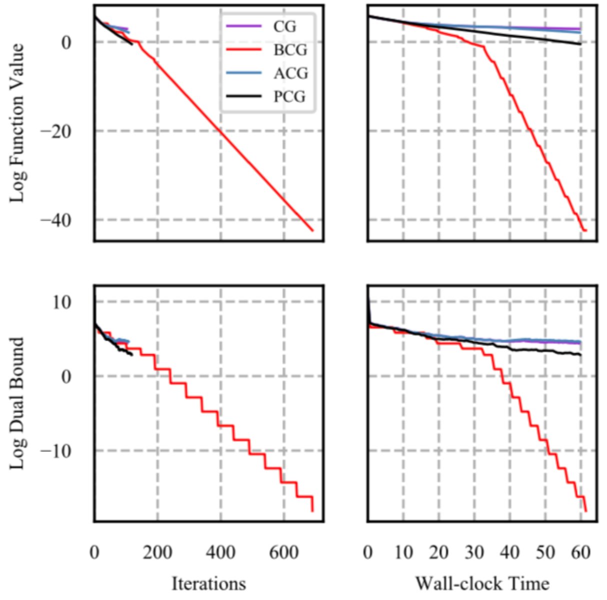
Lasso



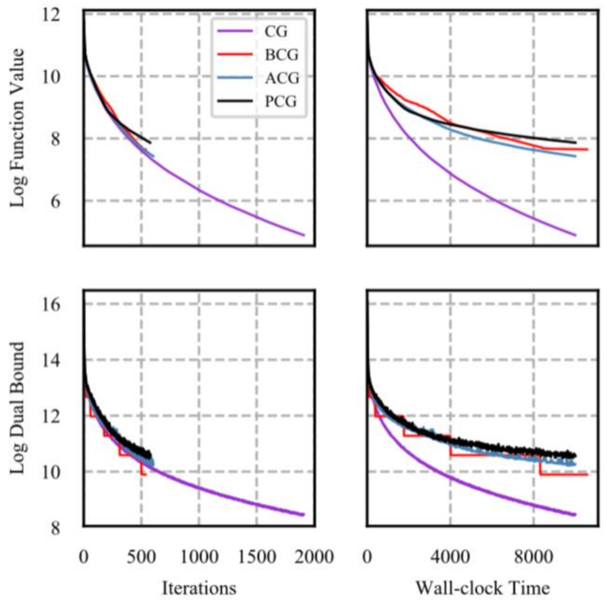
Structured Regression



Sparse Signal Recovery



Matrix Completion



Beyond Conditional Gradients: Blended Matching Pursuit

Georgia Tech Machine Learning

(Generalized) Matching Pursuit

Special variant of constraint convex minimization:

Given a set of vectors $\mathcal{A} \subseteq R^n$ solve:

$$\text{Min}_{\text{lin}(\mathcal{A})}\, f(x),$$

where f is a convex function. Basic example: $f_y(x) = ||x - y||^2$

Algorithm 1 (Orthogonal) Matching Pursuit

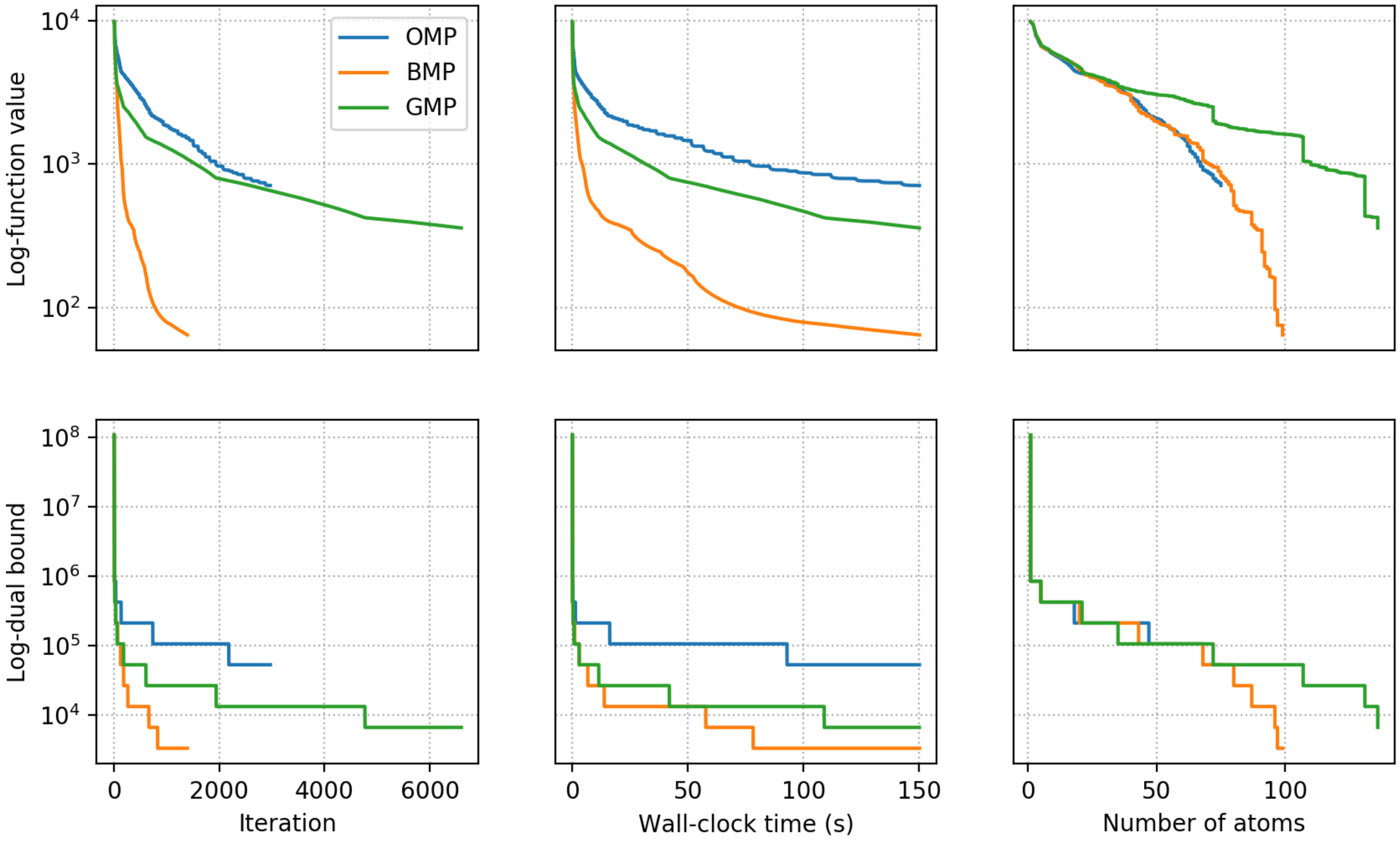
```
1: init $\mathbf{x}_0 \in \text{lin}(\mathcal{A})$ $\mathcal{S} = \{\mathbf{x}_0\}$
2: for $t = 0 \dots T$
3:     Find $\mathbf{z}_t := \text{(Approx-)LMO}_{\mathcal{A} \cup -\mathcal{A}}(-\mathbf{y} + \mathbf{x}_t)$
4:     $\mathcal{S} = \mathcal{S} \cup \mathbf{z}_t$
5:     Update MP: $\mathbf{x}_{t+1} := \arg\min_{\substack{\mathbf{x} := \mathbf{x}_t + \gamma \mathbf{z}_t \\ \gamma \in \mathbb{R}}} \|\mathbf{y} - \mathbf{x}\|^2$, or
6:     Update OMP: $\mathbf{x}_{t+1} := \arg\min_{\mathbf{x} \in \text{lin}(\mathcal{S})} \|\mathbf{y} - \mathbf{x}\|^2$
7: end for
```

Source: [Locatello et al '17]

Georgia Tech Machine Learning

Blended (Generalized) Matching Pursuit [Combettes, P. '19]
Faster than MP with sparsity of OMP

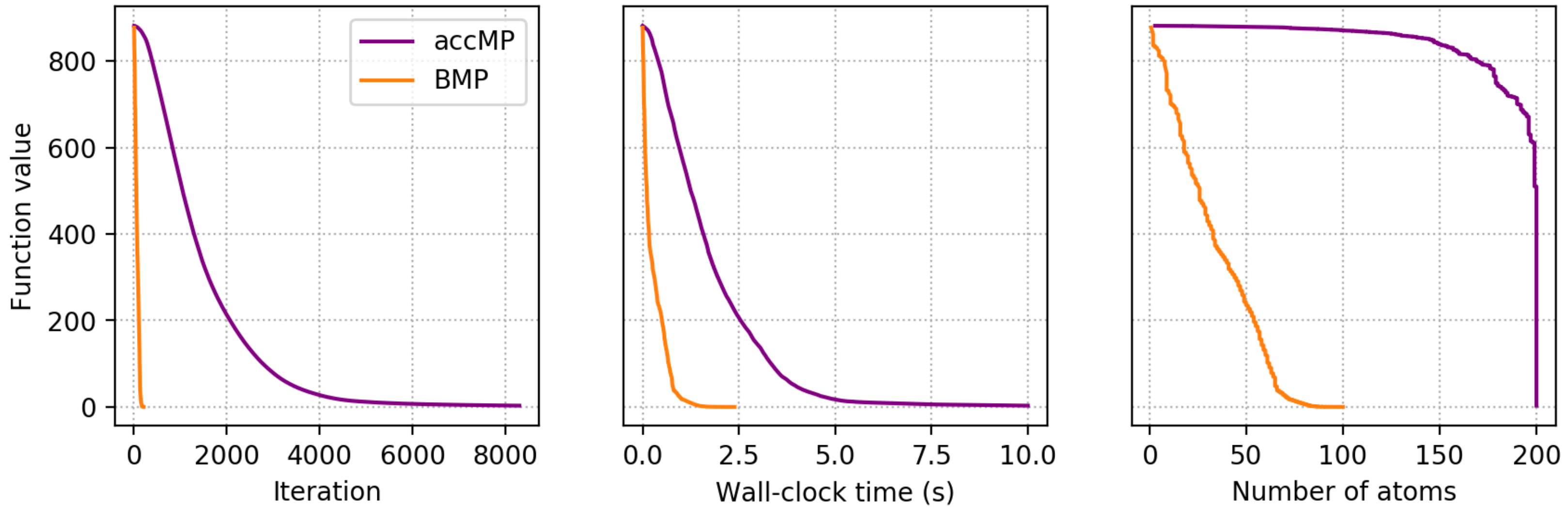
Inspired by [Locatello et al 2017] same blending idea can be used for MP.



Georgia Tech Machine Learning

Blended (Generalized) Matching Pursuit
Faster than MP with sparsity of OMP

Interestingly, (unaccelerated) BGMP even outperforms accelerated MP as using actual gradient directions over FW approximations seem to offset acceleration.

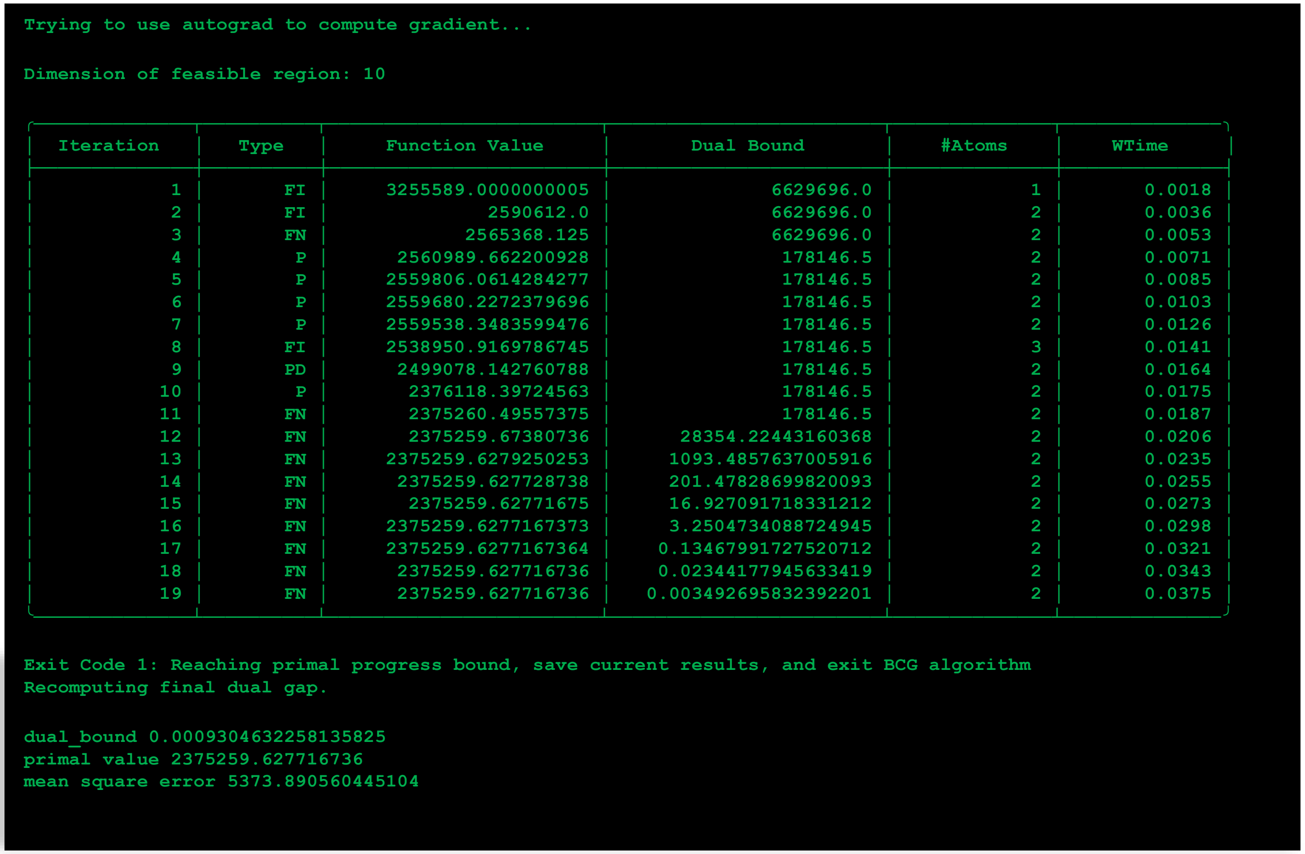


Georgia Tech Machine Learning

Code is publicly available

Code available at: https://www.github.com/pokutta/bcg

1. Python package for `Python 3.5+`
2. Reasonably well optimized (work in progress…)
3. Automatic differentiation via `autograd`
4. Interfaces with `gurobi`



```
Trying to use autograd to compute gradient...

Dimension of feasible region: 10

| Iteration |  Type |   Function Value   |      Dual Bound      |  #Atoms  |   WTime  |
|         1 |    FI | 3255589.0000000005 |            6629696.0 |        1 |   0.0018 |
|         2 |    FI |          2590612.0 |            6629696.0 |        2 |   0.0036 |
|         3 |    FN |        2565368.125 |            6629696.0 |        2 |   0.0053 |
|         4 |     P |  2560989.662200928 |             178146.5 |        2 |   0.0071 |
|         5 |     P | 2559806.0614284277 |             178146.5 |        2 |   0.0085 |
|         6 |     P | 2559680.2272379696 |             178146.5 |        2 |   0.0103 |
|         7 |     P | 2559538.3483599476 |             178146.5 |        2 |   0.0126 |
|         8 |    FI | 2538950.9169786745 |             178146.5 |        3 |   0.0141 |
|         9 |    PD |  2499078.142760788 |             178146.5 |        2 |   0.0164 |
|        10 |     P |   2376118.39724563 |             178146.5 |        2 |   0.0175 |
|        11 |    FN |   2375260.49557375 |             178146.5 |        2 |   0.0187 |
|        12 |    FN |   2375259.67380736 |    28354.22443160368 |        2 |   0.0206 |
|        13 |    FN | 2375259.6279250253 |   1093.4857637005916 |        2 |   0.0235 |
|        14 |    FN |  2375259.627728738 |   201.47828699820093 |        2 |   0.0255 |
|        15 |    FN |   2375259.62771675 |   16.927091718331212 |        2 |   0.0273 |
|        16 |    FN | 2375259.6277167373 |    3.2504734088724945 |        2 |   0.0298 |
|        17 |    FN | 2375259.6277167364 |   0.13467991727520712 |        2 |   0.0321 |
|        18 |    FN |  2375259.627716736 |  0.02344177945633419 |        2 |   0.0343 |
|        19 |    FN |  2375259.627716736 | 0.003492695832392201 |        2 |   0.0375 |

Exit Code 1: Reaching primal progress bound, save current results, and exit BCG algorithm
Recomputing final dual gap.

dual_bound 0.0009304632258135825
primal value 2375259.627716736
mean square error 5373.890560445104
```

Thank you for your attention!

Sebastian Pokutta

H. Milton Stewart School of Industrial and Systems Engineering
Center for Machine Learning @ GT (ML@GT)
Algorithm and Randomness Center (ARC)
Georgia Institute of Technology

Georgia Tech Machine Learning

twitter: @spokutta