Block Coordinate Descent (BCD) Methods/ Proximal Alernating Linearized (PALM) Method

Acknowledegement: part of the lecture slides by Yangyang Xu and Wotao Yin

References

- Y. Xu and W. Yin. A block coordinate descent method for regularized multi-convex optimization with applications to nonnegative tensor factorization and completion. SIAM Journal on imaging sciences, 6(3), pp. 1758–1789, 2013.
- Y. Xu and W. Yin. A globally convergent algorithm for nonconvex optimization based on block coordinate update. `http://arxiv.org/abs/1410.1386`
- Jerome Bolte, Shoham Sabach, Marc Teboulle, Proximal alternating linearized minimization for nonconvex and nonsmooth problems, mathematical programming

Outline

Regularized multi-convex optimization

Model

$$\min_x F(x_1, x_2, \ldots, x_s) = f(x_1, x_2, \ldots, x_s) + \sum_{i=1}^{s} r_i(x_i)$$

where

1. f is differentiable and multi-convex, generally non-convex; e.g. $f(x_1, x_2) = x_1^2 x_2^2 + 2x_1^2 + x_2$;

2. each r_i is convex, possibly non-smooth; e.g. $r_i(x_i) = \|x_i\|_1$;

3. r_i is defined on $\mathbb{R} \cup \infty$; it can enforce $x_i \in \mathcal{X}_i$ by setting

$$r_i(x_i) = \delta_{\mathcal{X}_i}(x_i) = \begin{cases} 0, & \text{if } x_i \in \mathcal{X}_i, \\ \infty, & \text{otherwise.} \end{cases}$$

Applications

- Low-rank matrix recovery (Recht et. al, 2010)

$$\min_{X,Y} \|\mathcal{A}(XY) - \mathcal{A}(M)\|^2 + \alpha\|X\|_F^2 + \beta\|Y\|_F^2$$

- Sparse dictionary learning (Mairal et. al, 2009)

$$\min_{D,X} \frac{1}{2}\|DX - Y\|_F^2 + \lambda \sum_i \|x_i\|_1, \text{ s.t. } \|d_j\|_2 \leq 1, \forall j;$$

- Blind source separation (Zibulevsky and Pearlmutter, 2001)

$$\min_{A,Y} \frac{1}{2}\|AYB - X\|_F^2 + \lambda\|Y\|_1, \text{ s.t. } \|a^j\|_2 \leq 1, \forall j;$$

Applications

- Nonnegative matrix factorization (Lee and Seung, 1999)

$$\min_{X,Y} \|M - XY\|_F^2, \text{ s.t. } X \geq 0; Y \geq 0;$$

- Nonnegative tensor factorization (Welling and Weber, 2001)

$$\min_{A_1,\ldots,A_N \geq 0} \|\mathcal{M} - A_1 \circ A_2 \circ \ldots \circ A_N\|_F^2;$$

Challenges

Non-convexity and non-smoothness cause

1. tricky convergence analysis;

2. expensive updates to all variables simultaneously.

Goal: to develop an efficient algorithm with simple update and global convergence (of course, to a stationary point)

Outline

Framework of block coordinate descent (BCD)

$$\min_x F(x_1, x_2, \ldots, x_s) = f(x_1, x_2, \ldots, x_s) + \sum_{i=1}^{s} r_i(x_i)$$

Algorithm 1: Block coordinate descent
Initialization:choose$(x_1^0, \ldots, x_s^0)$
for $k = 1, 2, \ldots,$ **do**
    for $i = 1, 2, \ldots, s$ **do**
        update x_i^k with all other blocks fixed
    end for
    if stopping criterion is satisfied **then**
    return $(x_1^k, \ldots, x_s^k)$
    end if
end for

Throughout iterations, each block x_i is updated by one of the three update schemes (coming next...)

Scheme 1: block minimization

The most-often used update:

$$x_i^k = \operatorname*{argmin}_{x_i} F(x_1^k, \ldots, x_{i-1}^k, x_i, x_{i+1}^{k-1}, \ldots, x_s^{k-1})$$

Existing results for differentiable convex F:

- Differentiable F and bounded level set $\Rightarrow$ objective converges to optimal value(Warga'63);
- Further with strict convexity $\Rightarrow$ sequence converges(Luo and Tseng'92);

Scheme 1: block minimization

Existing results for non-differentiable convex F:

- Non-differentiable F can cause stagnation at a non-critical point(Warga'63):
- Non-smooth part is separable $\Rightarrow$ subsequence convergence($i.e.$, exists a limit point) (Tseng'93)

Scheme 1: block minimization

Existing results for non-convex F:
May cycle or stagnate at a non-critical point (Powell'73):

$$F(x_1, x_2, x_3) = -x_1x_2 - x_2x_3 - x_3x_1 + \sum_{i=1}^{3}[(x_i - 1)_+^2 + (-x_i - 1)_+^2]$$

Each $F(x_i)$ has the form $(-a)x_i + [(x_i - 1)_+^2 + (-x_i - 1)_+^2]$

it minimizer $x_i^* = sign(a)(1 + 0.5\,|a|)$

Scheme 1: block minimization

Starting from $(-1-\epsilon, 1+\frac{1}{2}\epsilon, -1-\frac{1}{4}\epsilon)$ with $\epsilon \geq 0$, minimizing F over $x_1, x_2, x_3, x_1, x_2, x_3, \ldots$ produces:

$$\xrightarrow{x_1} (1+\tfrac{1}{8}\epsilon, 1+\tfrac{1}{2}\epsilon, -1-\tfrac{1}{4}\epsilon) \quad \xrightarrow{x_2} (1+\tfrac{1}{8}\epsilon, -1-\tfrac{1}{16}\epsilon, -1-\tfrac{1}{4}\epsilon)$$
$$\xrightarrow{x_3} (1+\tfrac{1}{8}\epsilon, -1-\tfrac{1}{16}\epsilon, 1+\tfrac{1}{32}\epsilon) \quad \xrightarrow{x_1} (-1-\tfrac{1}{64}\epsilon, -1-\tfrac{1}{16}\epsilon, 1+\tfrac{1}{32}\epsilon)$$
$$\xrightarrow{x_2} (-1-\tfrac{1}{64}\epsilon, 1+\tfrac{1}{128}\epsilon, 1+\tfrac{1}{32}\epsilon) \quad \xrightarrow{x_3} (-1-\tfrac{1}{64}\epsilon, 1+\tfrac{1}{128}\epsilon, -1-\tfrac{1}{256}\epsilon)$$

Scheme 1: block minimization

Remedies for non-convex F:

- F is differentiable and strictly quasiconvex over each block $\Rightarrow$ limit point is a critical point (Grippo and Sciandrone'00);

  quasiconvex: $F(\lambda x + (1-\lambda)y) \leq max(F(x), F(y)), \forall \lambda \in [0,1]$

- F is pseudoconvex over every two blocks and non-differentiable part is separable $\Rightarrow$ limit point is a critical point (Tseng'01);

  pseudoconvex: $\langle g, y-x \rangle \geq 0$,some $g \in \partial F(x) \Rightarrow F(x) \leq F(y)$

There is not global convergence result.

Scheme 2: block proximal descent

Adding $\|x_i - x_i^{k-1}\|_2^2$ gives better stability:

$$x_i^k = \underset{x_i}{\operatorname{argmin}}\, F(x_1^k, \ldots, x_{i-1}^k, x_i, x_{i+1}^{k-1}, \ldots, x_s^{k-1}) + \frac{L_i^{k-1}}{2}\|x_i - x_i^{k-1}\|^2;$$

Convergence results require fewer assumptions on F:

- F is convex $\Rightarrow$ objective converges to optimal value (Auslender'92);
- F is non-convex $\Rightarrow$ limit point is stationary (Grippo and Sciandrone'00);

Non-smooth terms must still be separable. No global convergence for non-convex F.

Scheme 3: block proximal linear

Linearize f over block i and add $\frac{L_i^{k-1}}{2}\|x_i - \hat{x}_i^{k-1}\|^2$:

$$x_i^k = \underset{x_i}{\operatorname{argmin}} \left\langle \nabla_i f, x_i - \hat{x}_i^{k-1} \right\rangle + r_i(x_i) + \frac{L_i^{k-1}}{2}\|x_i - \hat{x}_i^{k-1}\|^2;$$

where $f = f(x_1^k, \ldots, x_{i-1}^k, x_i, x_{i+1}^{k-1}, \ldots, x_s^{k-1})$

Scheme 3: block proximal linear

- Extrapolate $\hat{x}_i^{k-1} = x_i^{k-1} + w_i^{k-1}(x_i^{k-1} - x_i^{k-2})$ with weight $w_i^{k-1} \geq 0$
- Much easier than schemes 1 & 2; may have closed-form solutions for simple r_i;
- Used in randomized BCD for differentiable convex problems (Nesterov'12);
- The update is less greedy than schemes 1 & 2, causes more iterations, but may save total time;
- Empirically, the "relaxation" tend to avoid "shallow-puddle" local minima better than schemes 1 & 2 .

Comparisons

1. Block coordinate minimization (scheme 1) is mostly used
   - May generally cycle or stagnate at a non-critical point (Powell'73);
   - Globally convergent for strictly convex problem (Luo and Tseng'92);
   - For non-convex problem, each limit point is a critical point if each subproblem has unique solution and objective is regular (Tseng'01);
   - Global convergence for non-convex problems is unknown;

Comparisons

2. Block proximal (scheme 2) can stabilize iterates
   - Each limit point is a critical point (Grippo and Sciandrone'00);
   - Global convergence for non-convex problems is unknown;

3. Block proximal linearization (scheme 3) is often easiest
   - Very few works use this scheme for non-convex problems yet;
   - Related to the coordinate gradient descent method (Tseng and Yun'09).

Why different update schemes?

- They deal with subproblems of different properties;
- Implementations are easier for many applications;
- Schemes 2 & 3 may save total time than scheme 1;
- Convergence can be analyzed in a unified way.

Example: sparse dictionary learning

$$\min_{D,X} \frac{1}{2}\|DX - Y\|_F^2 + \lambda \sum_i \|x_i\|_1, \text{ s.t. } \|D\|_F \leq 1$$

apply scheme 1 to D and scheme 3 to X; both are closed-form.

Examples of global convergence by BCD

- Low-rank matrix recovery (Recht et. al, 2010)

$$\min_{X,Y} \|\mathcal{A}(XY) - \mathcal{A}(M)\|^2 + \alpha\|X\|_F^2 + \beta\|Y\|_F^2$$

- Sparse dictionary learning (Mairal et. al, 2009)

$$\min_{D,X} \frac{1}{2}\|DX - Y\|_F^2 + \|X\|_1 + \delta_{\mathcal{D}}(D), \mathcal{D} = \{D : \|d_j\|_2 \leq 1, \forall j\}$$

- Blind source separation (Zibulevsky and Pearlmutter, 2001)

$$\min_{A,Y} \frac{\lambda}{2}\|AYB - X\|_F^2 + \|Y\|_1 + \delta_{\mathcal{A}}(A), \mathcal{A} = \{A : \|a^j\|_2^2 \leq 1, \forall j\}$$

- Nonnegative matrix factorization (Lee and Seung, 1999)

$$\min_{X,Y} \|M - XY\|_F^2, \textit{subject to } X \geq 0; Y \geq 0;$$

Outline

Framework of Bolte, Sabach, Teboulle

Consider the model:

$$(M) \qquad \min_{x,y} \quad \Psi(x,y) = f(x) + g(y) + H(x,y)$$

Let

$$\text{prox}_t^{\sigma}(x) = \arg\min_u \ \sigma(u) + \frac{t}{2}\|u - x\|^2$$

A single iteration of PALM:

- Take $\gamma_1 > 1$, set $c_k = \gamma_1 L_1(y^k)$ and compute

$$x^{k+1} = \text{prox}^f_{c_k}(x^k - \frac{1}{c_k}\nabla_x H(x^k, y^k))$$

- Take $\gamma_2 > 1$, set $d_k = \gamma_2 L_2(x^{k+1})$ and compute

$$y^{k+1} = \text{prox}^g_{d_k}(y^k - \frac{1}{d_k}\nabla_y H(x^{k+1}, y^k))$$

Nonconvex Proximal Operator

Let

$$m^{\sigma}(x,t) = \inf_{u} \ \sigma(u) + \frac{1}{2t}\|u-x\|^2$$

Well-definedness of proximal maps:

- Let $\sigma(u)$ be a proper and lower semicontinuous function with $\inf \ \sigma > -\infty$. Then, for every $t \in (0,\infty)$, the set $\operatorname{prox}^{\sigma}_{1/t}(x)$ is nonempty and compact. In addition, $m^{\sigma}(x,t)$ is finite and continuous in (x,t).
- When $\sigma = \delta_X$, the indicator function of a nonempty and closed set X, the proximal map reduces to the projection operator.

Subdifferentials of nonconvex and nonsmooth fun

$\sigma(u) : \mathbb{R}^d \to (-\infty, +\infty)$ be a proper and lower semicontinuous fun.

- For a given $x \in \mathbf{dom}\ \sigma$, the Fréchet subdifferential of σ at x, written $\hat{\partial}\sigma(x)$, is the set of all vectors $u \in \mathbb{R}^d$ which satisfy

$$\liminf_{y \neq x \; y \to x} \frac{\sigma(y) - \sigma(x) - \langle u, y - x \rangle}{\|y - x\|} \geq 0$$

- The limiting-subdifferential, or simply the subdifferential, of σ at $x \in \mathbb{R}^n$, written $\partial\sigma(x)$, is defined through the following closure process

$$\partial\sigma(x) = \{u \in \mathbb{R}^d : \exists x^k \to x, \sigma(x^k) \to \sigma(x), \text{ and } u^k \in \hat{\partial}\sigma(x^k) \to u, k \to \infty\}$$

- Assume that the coupling function H in Problem (M) is continuously differentiable. Then for all $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$:

$$\partial\Psi(x, y) = (\nabla_x H(x, y) + \partial f(x), \nabla_y H(x, y) + \partial g(y))$$

Assumptions

- $\inf \Psi > -\infty$, $\inf f > -\infty$ and $\inf g > -\infty$
- For any fixed y, the partial gradient $\nabla_x H(x,y)$ is Lipschitz with moduli $L_1(y)$. For any fixed x, $\nabla_y H(x,y)$ is Lipschitz with moduli $L_2(x)$
- There exists $\lambda_i^-, \lambda_i^+ > 0$ such that

$$\inf\{L_1(y^k) : k \in N\} \geq \lambda_1^- \text{ and } \inf\{L_2(x^k) : k \in N\} \geq \lambda_2^-$$
$$\sup\{L_1(y^k) : k \in N\} \leq \lambda_1^+ \text{ and } \sup\{L_2(x^k) : k \in N\} \leq \lambda_2^+$$

- ∇H is Lipschitz continuous on bounded subsets of $\mathbb{R}^n \times \mathbb{R}^m$:

$$\|(\nabla_x H(x_1,y_1) - \nabla_x H(x_2,y_2), \nabla_y H(x_1,y_1) - \nabla_y H(x_2,y_2))\| \leq M\|(x_1 - x_2, y_1 - y_2)\|$$

Informal Proofs

- Sufficient decrease property: Find a positive constant ρ_1 such that
$$\rho_1 \|z^{k+1} - z^k\|^2 \leq \Psi(z^k) - \Psi(z^{k+1})$$
- A subgradient lower bound for the iterates gap: Assume that the sequence generated by the algorithm is bounded. Find another positive constant ρ_2, such that
$$\|w^{k+1}\| \leq \rho_2 \|z^{k+1} - z^k\|, \quad w^k \in \partial\Psi(z^k)$$
- Using the KL property: Assume that Ψ is a KL function and show that the generated sequence $\{z^k\}_{k\in N}$ is a Cauchy sequence.

Note that when the first two properties hold, then for any algorithm one can show that the set of accumulations points is a nonempty, compact and connected set.

The Kurdyka-Łojasiewicz (KL) property

- For any subset $S \subset \mathbb{R}^d$ and $x \in \mathbb{R}^d$, $\mathrm{dist}(x, S) = \inf_y \ \|y - x\|$.
- Let $\eta \in (0, +\infty)$. Φ_η is the class of all concave and continuous functions $\psi : [0, \eta) \to \mathbb{R}_+$: (i) $\psi(0) = 0$, (ii) ψ is C^1 on $(0, \eta)$ and continuous at 0; (iii) for all $s \in (0, \eta)$, $\psi'(s) > 0$.
- Let σ be proper and lower semicontinuous.
- σ has KL property at $\bar{u} \in \mathbf{dom}\ \partial\sigma := \{u \in \mathbb{R}^d \mid \partial\sigma(u) \neq \emptyset\}$: if there exits $\eta \in (0, +\infty)$, a neighborhood U of $\bar{u}$ and a function $\psi \in \Phi_\eta$ such that for all $u \in U \cap [\sigma(\bar{u}) < \sigma(u) < \sigma(\bar{u}) + \eta]$, it holds:

$$\psi'(\sigma(u) - \sigma(\bar{u}))\mathrm{dist}(0, \partial\sigma(u)) \geq 1$$

- If σ satisfy the KL property at each point of $\mathbf{dom}\ \partial\sigma$, then σ is called a KL function

The Kurdyka-Łojasiewicz (KL) sets and functions

semi-algebraic, subanalytic and log-exp are KL functions

- A subset S of $\mathbb{R}^d$ is a real semi-algebraic set if there exists a finite number of real polynomial functions $g_{ij}, h_{ij} : \mathbb{R}^d \to \mathbb{R}$ such that

$$S = \cup_{j=1}^{p} \cap_{i=1}^{q} \{u \in \mathbb{R}^d : g_{ij}(u) = 0, h_{ij}(u) < 0\}$$

- A function $h : \mathbb{R}^d \to (-\infty, +\infty]$ is called semi-algebraic if its graph $\{(u,t) \in \mathbb{R}^{d+1} : h(u) = t\}$ is a semi-algebraic subset of $\mathbb{R}^{d+1}$
- Let $\sigma(u) : \mathbb{R}^d \to (-\infty, +\infty)$ be a proper and lower semicontinuous function. If σ is semi-algebraic then it satisfies the KL property at any point of $\mathbf{dom}\sigma$.

The Kurdyka-Łojasiewicz (KL) sets and functions

Examples:

- Real polynomial functions.
- Indicator functions of semi-algebraic sets.
- Finite sums and product of semi-algebraic functions.
- Composition of semi-algebraic functions.
- Sup/Inf type function, e.g., $\sup\{g(u,v) : v \in C\}$ is semi-algebraic when g is a semi-algebraic function and C a semi-algebraic set.
- In matrix theory, all the following are semi-algebraic sets: cone of PSD matrices, Stiefel manifolds and constant rank matrices.
- The function $x \to \operatorname{dist}(x,S)^2$ is semi-algebraic whenever S is a nonempty semialgebraic subset of $\mathbb{R}^d$.
- $\|\cdot\|_0$, $\|\cdot\|_p$ with a rational p are semi-algebraic

Proofs: Sufficient decrease property

- Let $h : \mathbb{R}^d \to R$ be a continuously differentiable function with gradient ∇h assumed L_h-Lipschitz continuous and let $\sigma : \mathbb{R}^d \to (-\infty, +\infty]$ be a proper and lower semicontinuous function with $\inf_{\mathbb{R}^d} \sigma > -\infty$. Fix any $t > L_h$. Then, for any $u \in \mathbf{dom}\sigma$ and any

$$u^+ \in \mathrm{prox}_t^\sigma (u - \frac{1}{t}\nabla h(u)),$$

we have

$$h(u^+) + \sigma(u^+) \leq h(u) + \sigma(u) - \frac{1}{2}(t - L_h)\|u - u^+\|^2$$

Proofs: convergence property

- The sequence $\{\Psi(z^k)\}_{k\in N}$ is nonincreasing and with $\rho_1 = \min\{(\gamma_1 - 1)\lambda_1^-, (\gamma_2 - 1)\lambda_2^-\}$:

$$\frac{\rho_1}{2}\|z^{k+1} - z^k\|^2 \leq \Psi(z^k) - \Psi(z^{k+1}), \forall k \geq 0 \tag{1}$$

- We have

$$\sum_{k=1}^{\infty} \|x^{k+1} - x^k\|^2 + \|y^{k+1} - y^k\|^2 = \sum_{k=1}^{\infty} \|z^{k+1} - z^k\|^2 < \infty$$

and hence $\lim_{k\to\infty} \|z^{k+1} - z^k\| = 0$

Proofs: subgradient lower bound for the iterates gap

- Define $\rho_2 = \max\{\gamma_1\lambda_1^+, \gamma_2\lambda_2^+\}$ and

$$\begin{aligned} A_x^k &= c_{k-1}(x^{k-1} - x^k) + \nabla_x H(x^k, y^k) - \nabla_x H(x^{k-1}, y^{k-1}) \\ A_y^k &= d_{k-1}(y^{k-1} - y^k) + \nabla_y H(x^k, y^k) - \nabla_y H(x^k, y^{k-1}) \end{aligned}$$

Then $(A_x^k, A_y^k) \in \partial\Psi(x^k, y^k)$ and there exists $M > 0$:

$$\|(A_x^k, A_y^k)\| \leq \|A_x^k\| + \|A_y^k\| \leq (2M + 3\rho_2)\|z^k - z^{k-1}\|$$

Proofs: Uniformized KL property

- Let Ω be compact and σ be proper and lower semicontinuous. Assume σ is constant on Ω and satisfy the KL property at each point of Ω. Then, $\exists \epsilon > 0, \eta > 0$ and $\psi \in \Psi_\eta$ such that for $\bar{u} \in \Omega$,

$$u \in \{u \in \mathbb{R}^d : \mathrm{dist}(u, \Omega) < \epsilon\} \cap [\sigma(\bar{u}) < \sigma(u) < \sigma(\bar{u}) + \eta]$$

  one has

$$\psi'(\sigma(u) - \sigma(\bar{u}))\mathrm{dist}(0, \partial\sigma(u)) \geq 1$$

Using KL Property

- for sufficiently large k

$$\psi'(\Psi(z^k) - \Psi(\bar{z}))\mathrm{dist}(0, \partial\Psi(z^k)) \geq 1$$

- subgradient lower bound for the iterates gap:

$$\psi'(\Psi(z^k) - \Psi(\bar{z})) \geq \frac{1}{2M + 3\rho_2}\|z^k - z^{k-1}\|^{-1}$$

- the concavity of ψ gives:

$$\psi(\Psi(z^k)-\Psi(\bar{z}))-\psi(\Psi(z^{k+1})-\Psi(\bar{z})) \geq \psi'(\Psi(z^k)-\Psi(\bar{z}))(\Psi(z^k)-\Psi(z^{k+1}))$$

- define:

$$\Delta_{p,q} := \psi(\Psi(z^p) - \Psi(\bar{z})) - \psi(\Psi(z^q) - \Psi(\bar{z}))$$

- Together with (1):

$$\Delta_{k,k+1} \geq \frac{\|z^{k+1} - z^k\|^2}{C\|z^k - z^{k-1}\|}$$

Using KL Property

- Establish the inequality:

$$\sum_{i=l+1}^{k} \|z^{i+1} - z^i\| \leq \|z^{l+1} - z^l\| + C\Delta_{l+1,k+1}$$

- Since $\psi \geq 0$, we have:

$$\sum_{i=l+1}^{k} \|z^{i+1} - z^i\| \leq \|z^{l+1} - z^l\| + C\psi(\Psi(z^{l+1}) - \Psi(\bar{z}))$$

- Therefore:

$$\sum_{k=1}^{\infty} \|z^{k+1} - z^k\| < \infty$$

Convergence of PALM to critical points

Suppose that Ψ is a KL function. Let $\{z^k\}_{k\in N}$ be a sequence generated by PALM which is assumed to be bounded.

- The sequence $\{z^k\}_{k\in N}$ has finite length:

$$\sum_{k=1}^{\infty} \|z^{k+1} - z^k\| < \infty$$

- The sequence $\{z^k\}_{k\in N}$ converges to a critical point $z^* = (x^*, y^*)$ of Ψ

Extension of PALM for problems with $p \geq 1$ blocks

Convergence of PALM to critical points

Choose $\psi(s) = cs^{1-\theta}$, where $c > 0$ and $\theta \in [0, 1)$.

- If $\theta = 0$, then the sequence converges in a finite number of steps
- If $\theta \in (0, 1/2]$, then there exists $\omega > 0$ and $\tau \in [0, 1)$ such that $\|z^k - \bar{z}\| \leq \omega\tau^k$
- If $\theta \in (1/2, 1)$, then there exits $\omega > 0$ such that

$$\|z^k - \bar{z}\| \leq wk^{-\frac{1-\theta}{2\theta-1}}$$

Outline

Expressing $X \succ 0$ by Schur complement

- Assume $X \in S^n$ is partitioned as $\begin{pmatrix} \xi & y^\top \\ y & B \end{pmatrix}$, where $\xi \in \mathbb{R}$, $y \in \mathbb{R}^{n-1}$ and $B \in S^{n-1}$ is nonsingular
- Factorization:

$$X = \begin{pmatrix} 1 & y^\top B^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} \xi - y^\top B^{-1} y & 0 \\ 0 & B \end{pmatrix} \begin{pmatrix} 1 & 0 \\ B^{-1} y & I \end{pmatrix}$$

Positive definiteness and Schur complement:

$$X \succ 0 \iff B \succ 0 \text{ and } (X/B) := \xi - y^\top B^{-1} y > 0$$

- Cholesky factorization: $B := LL^\top$.
- $\xi - y^\top B^{-1} y > 0 \iff \|L^{-1} y\|^2 \leq \xi$ (second-order cone)

Constructing SOC constraint row by row

- Given $X^k \succ 0$, we can fix the principal submatrix

$$B := X^k_{1^c,1^c} = \begin{pmatrix} X^k_{2,2} & \cdots & X^k_{2,\,n} \\ \cdots & \cdots & \cdots \\ X^k_{n,\,2} & \cdots & X^k_{n,n} \end{pmatrix}$$

  and let $\xi := X_{1,1}$ and $y := X_{1^c,1} := (X_{1,2}, \cdots, X_{1,n})^\top$
- The variable X now is $\begin{pmatrix} \xi & y^\top \\ y & B \end{pmatrix} := \begin{pmatrix} \xi & y^\top \\ y & X^k_{1^c,1^c} \end{pmatrix}$
- SOC constraint: $\xi - y^\top B^{-1} y \geq \nu$ for $\nu > 0$
- In general: $\xi := X_{i,i}$, $y := X_{i^c,i}$ and $B := X^k_{i^c,i^c}$

Solving RBR subproblem

SDP			SOCP restriction	
$\min\limits_{X \in S^n}$	$\langle C, X \rangle$		$\min\limits_{[\xi;y] \in \mathbb{R}^n}$	$\widetilde{c}^\top [\xi; y]$
s.t.	$\mathcal{A}(X) = b,$	$\Longrightarrow$	s.t.	$\widetilde{A}\,[\xi; y] = \widetilde{b},$
	$X \succeq 0,$			$\xi - y^\top B^{-1} y \geq \nu,$

where $\nu > 0$ and

$$\widetilde{c} := \begin{pmatrix} C_{i,i} \\ 2C_{i^c,i} \end{pmatrix}, \quad \widetilde{A} := \begin{pmatrix} A^{(1)}_{i,i} & 2A^{(1)}_{i,i^c} \\ \cdots & \cdots \\ A^{(m)}_{i,i} & 2A^{(m)}_{i,i^c} \end{pmatrix} \text{ and } \widetilde{b} := \begin{pmatrix} b_1 - \left\langle A^{(1)}_{i^c,i^c}, B \right\rangle \\ \cdots \\ b_m - \left\langle A^{(m)}_{i^c,i^c}, B \right\rangle \end{pmatrix}$$

Row-by-Row (RBR) algorithm prototype

Algorithm 1: A row-by-row (RBR) method prototype

Set $X^1 \succ 0$, $\nu \geq 0$ and $k := 1$.;
while *not converge* **do**
    for $i = 1, \cdots, n$ **do**
        Solve the SOCP subproblem for i-th row/column.;
        Update $X^k_{i,i} := \xi$, $X^k_{i^c,i} := y$ and $X^k_{i,i^c} := y^\top$.
    Set $X^{k+1} := X^k$ and $k := k + 1$.

Application: the maxcut SDP relaxation

The RBR subproblem for SDP with only diagonal element constraints:

$$\begin{array}{ll} \min\limits_{X\in S^n} & \langle C, X\rangle \\ \text{s.t.} & X_{i,i} = 1, \\ & X \succeq 0, \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \min\limits_{y\in\mathbb{R}^{n-1}} & \widehat{c}^{\top} y \\ \text{s.t.} & 1 - y^{\top} B^{-1} y \geq \nu \end{array}$$

Closed-form solution of the RBR subproblem

If $\gamma := \widehat{c}^{\top} B \widehat{c} > 0$, the solution of the RBR subproblem is

$$y = -\sqrt{\frac{1-\nu}{\gamma}} B\widehat{c}.$$

Otherwise, the solution is $y = 0$.

Interpretation in terms of log-barrier approach

- Consider the logarithmic barrier problem

$$\begin{aligned} \min_{X \in S^n} \quad & \langle C, X \rangle - \sigma \log\det X \\ \text{s.t.} \quad & X_{ii} = 1, \forall i = 1, \cdots, n, \quad X \succeq 0 \end{aligned}$$

- Key: $\det(X) = \det(B)(1 - y^\top B^{-1} y)$
- The RBR subproblem is:

$$\min_{y \in \mathbb{R}^{n-1}} \quad \widehat{c}^\top y - \sigma \log(1 - y^\top B^{-1} y)$$

  whose solution is $y = -\frac{\sqrt{\sigma^2+\gamma}-\sigma}{\gamma} B\widehat{c}$, where $\gamma := \widehat{c}^\top B \widehat{c}$.

- Equal to the pure RBR method if $\nu = 2\sigma \frac{\sqrt{\sigma^2+\gamma}-\sigma}{\gamma}$

Convergence for general function

Consider the RBR method for solving

$$(P) \qquad \begin{array}{ll} \min\limits_{X \in S^n} & f(X) - \sigma \log\det X \\ \text{s.t.} & L \le X \le U, \quad X \succ 0 \end{array}$$

- $f(X)$ is a convex function of X
- $L, U \in S^n$ are constant matrices and $L \le X \le U$ means that $L_{i,j} \le X_{i,j} \le U_{i,j}$ for all $i, j = 1, \cdots, n$

Theorem 1.

Let $\{X^k\}$ be a sequence generated by the row-by-row method for solving (P). Then every limit point of $\{X^k\}$ is a global minimizer of (P).

Failure on a SDP with general linear constraints

Consider the SDP

$$\begin{array}{ll} \min & X_{11} + X_{22} - \log\det(X) \\ \text{s.t.} & X_{11} + X_{22} \geq 4, \quad X \succeq 0. \end{array}$$

- Initial point: $\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$ and optimal solution: $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$.
- The RBR subproblems are

$$\begin{array}{ll} \min & X_{11} - \log(3X_{11} - X_{12}^2), \text{ s.t. } X_{11} \geq 1, \\ \min & X_{22} - \log(X_{22} - X_{12}^2), \quad \text{s.t. } X_{22} \geq 3 \end{array}$$

- Optimal solutions of subproblems are, respectively, $X_{11} = 1$, $X_{12} = 0$ and $X_{12} = 0$, $X_{22} = 3$

Outline

Stochastic Gradient Descent

Consider

$$\min_{x \in \mathbb{R}^n} f(x) = \sum_{e \in E} f_e(x_e)$$

- e denotes a small subset of $\{1, 2, \ldots, n\}$.
- In many of machine learning problems, n and $|E|$ are both large. **Note**: the 'set' E can contain a few copies of the same element e.
- f_e only acts on a few components of x, say x_e.
- f_e can easily be regarded as a function over $\mathbb{R}^n$, just by ignoring the components not in the subset e.

Problem Definition

Example: machine learning applications

- Minimize the empirical risk

$$\min_x \frac{1}{n} \sum_{i=1}^{n} l_i(a_i^T x)$$

- a_i represents the ith data point, x is the model. l_i is a loss function.
- Logistic regression, least squares, SVM ...
- If each a_i is sparse, then it becomes our problem today.

Problem Definition

Example: generic minimization problem

- Minimize the following function

$$\min_{x_1,\ldots,x_{m_1}} \min_{y_1,\ldots,y_{m_2}} \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \phi_{i,j}(x_i, y_j)$$

- $\phi_{i,j}$ is a convex scalar function. x_i and y_j are vectors.
- For matrix completion and matrix factorization:

$$\phi_{i,j} = (A_{i,j} - x_i^T y_j)^2$$

- $n = m_1 m_2$ functions, each of which interacts with only two variables.
- A variable is shared among at most $m_1 + m_2$ functions.

Stochastic Gradient Descent

- Gradient method is given by

$$x_{k+1} = x_k - \gamma_k \nabla f(x_k)$$

where $\nabla f(x_k) = \sum_{e \in E} \nabla f_e(x_k)$.

- Stochastic Gradient Descent(SGD)

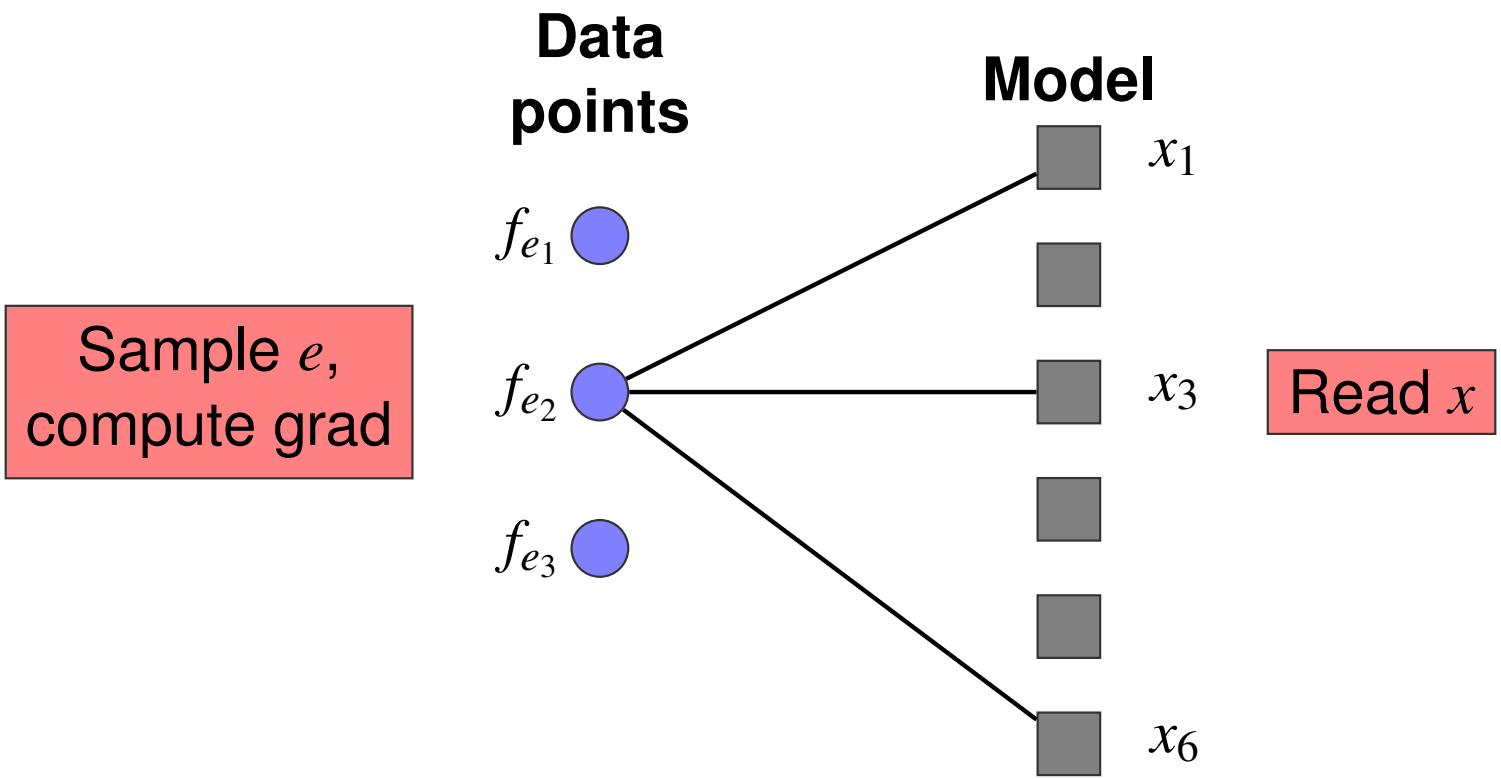
$$x_{k+1} = x_k - \gamma_k \nabla f_{s_k}(x_k)$$

where s_k is randomly sampled from E.

- γ_k is the step size(or learning rate). Can be a constant or shrinking.
- Idea of SGD: computing the entire $\nabla f(x)$ may be too costly for large $|E|$ and n. Can we compute just a small proportion of $\nabla f(x)$ and ensure the convergence?

Stochastic Gradient Descent

One step of SGD:



SGD: Advantages

SGD has been a round for a while, for good reasons:

- Less computation than classic GD.
- Robust to noise.
- Simple to implement.
- Near-optimal learning performance.
- Small computational foot-print.
- Stable.

Parallel SGD

- It may takes a **HUGE** number of updates for SGD on large datasets.
- **Goal**: a parallel version of SGD.
- **How to parallelize SGD?**
  - Parallelize one update – computing $\nabla f_{s_k}(x_k)$ is cheap, even for deep nets. Thus it may be not worth working out parallel codes on a cheap subroutine.
  - Parallelize the updates – SGD is sequential, making it nearly impossible for the parallel stuff.
  - Can we parallelize a sequential algorithm?

Parallel SGD

Can we parallelize a sequential algorithm?

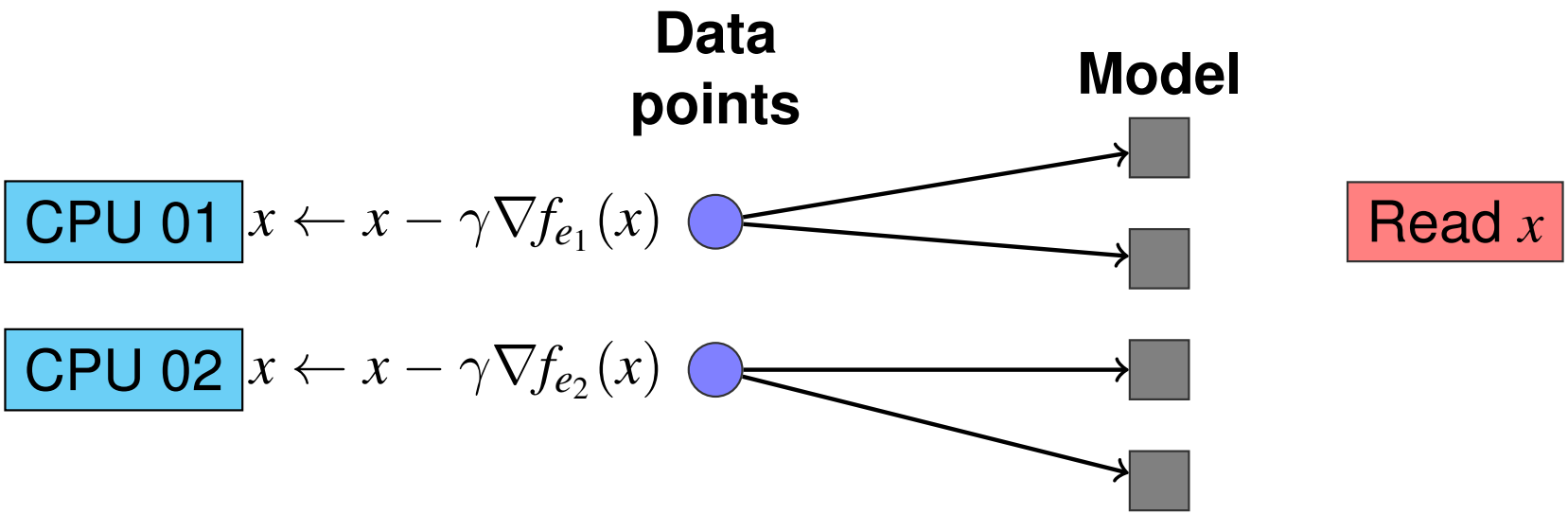
- No – for most cases.
- Almost yes – for problems with structures of sparsity.
- For our problem:

$$\min_{x \in \mathbb{R}^n} f(x) = \sum_{e \in E} f_e(x_e)$$

  If most of f_e's don't share the same component of x, may be we can exploit the sparsity to parallelize SGD.

Parallel SGD

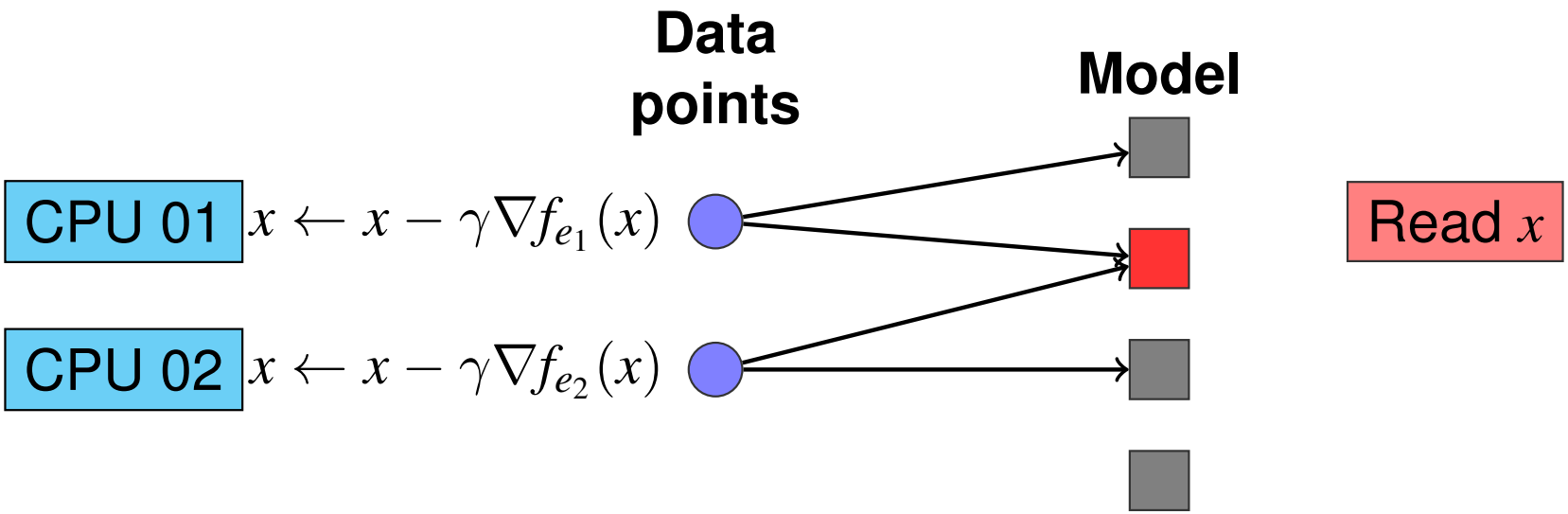
Good Case:



- The components of f_{e_1} and f_{e_2} don't overlap.
- Two parallel updates means two serial updates.
- No conflicts means speed up!

Parallel SGD

Bad Case:



- The components of f_{e_1} and f_{e_2} overlap at x_2.
- Conflicts mean less parallelization.
- What should be done to handle the conflicts?

Why conflicts affect so much?

- CPUs don't have direct access to the memory.
- Computations can only be done on CPU cache.
- Data are read from the memory to CPU cache. After the computations are finished, results are pushed back to the memory.
- A CPU has no idea whether its 'colleagues' have local updates which have not been pushed to the memory.

Parallel SGD

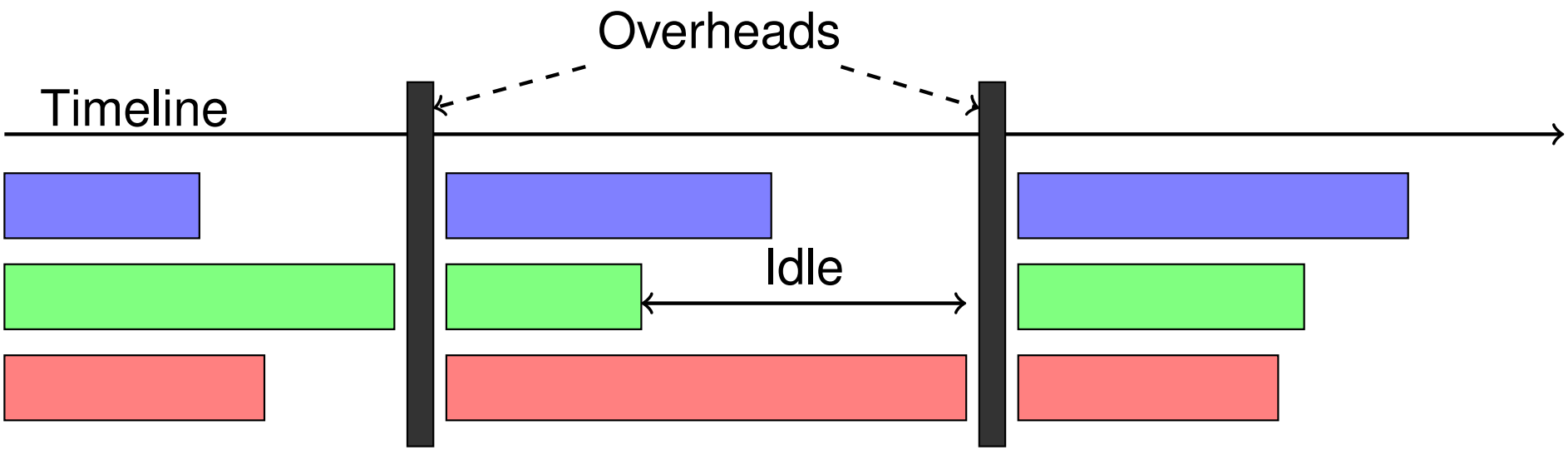
How to deal with the conflicts?

- By ‘coordinate’ or ‘memory lock’ approach.
- **Lock type**: exclusive lock.
- Tell others: I’m updating the variable NOW. Please don’t make any changes until I have finished.
- Ensure the correctness when performing parallel SGD.
- Provide only limited speedup for making parallel SGD as a ‘sequential’ program. Things are even worse when conflicts occur too often – even slower than the sequential version of SGD!

Asynchronous Updates

Q: How to deal with the conflicts?
A: Asynchronous programming tells us to just ignore it.

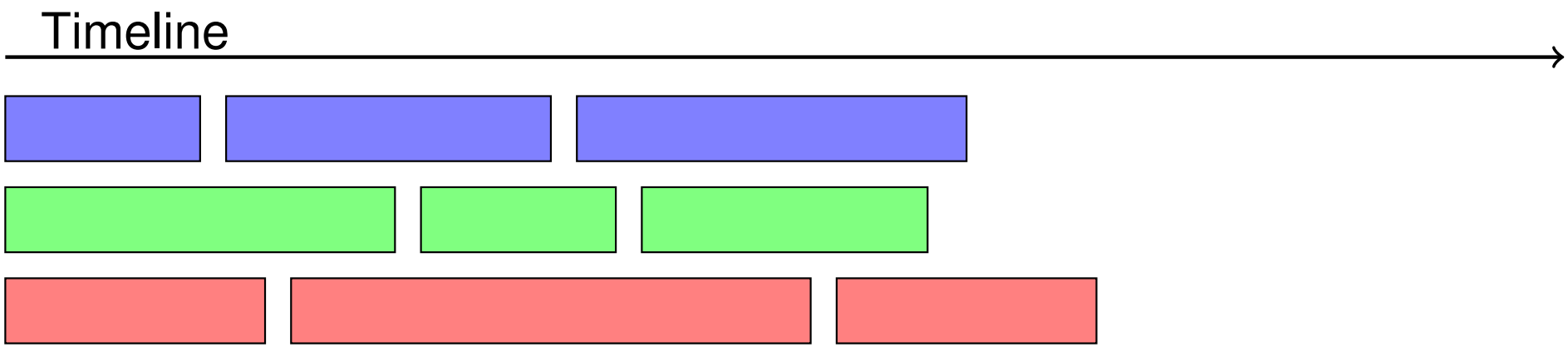
The synchronous world:



- Load imbalance causes the idle.
- Correct but slow.

Asynchronous Updates

The asynchronous world:



- No synchronizations among the workers.
- No idle time – every worker is kept busy.
- High scalability.
- Noisy but fast.

Hogwild! the Asynchronous SGD

If we remove all locking and synchronizing processes in parallel SGD. Then we obtain the Hogwild! algorithm.

Algorithm 1 Hogwild! the Asynchronous SGD

Each processor asynchronously do
loop
    Sample e randomly from E.
    Read the current point x_e from the memory.
    Compute $G_e(x) := |E|\nabla f_e(x)$.
    for $v \in e$ **do**
        $x_v \leftarrow x_v - \gamma b_v^T G_e(x)$.
    end for
end loop

Hogwild! the Asynchronous SGD

A variant of Hogwild!:

Algorithm 2 Hogwild! the Asynchronous SGD(Variant 1)

1: Each processor asynchronously do
2: **loop**
3: Sample e randomly from E.
4: Read the current point x_e from the memory.
5: Compute $G_e(x) := |E|\nabla f_e(x)$.
6: Sample $v \in e$, then $x_v \leftarrow x_v - \gamma |e| b_v^T G_e(x)$.
7: **end loop**

Note:

- The entire gradient is computed. Only one component is updated.
- The $|e|$ factor ensures $\mathbb{E}[|e| b_v^T G_e(x)] = \nabla f(x)$.
- Seems wasteful, but easy to analyze.

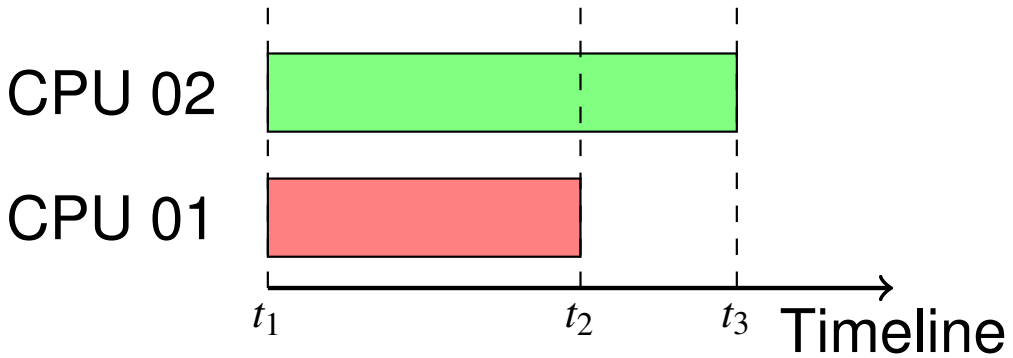
Issues of the Asynchronous Updates

- The inconsistency between the CPU cache and the memory makes the results incorrect.
- Older updates can be overwritten by newer ones.
- Suppose we want to perform two updates with two threads

$$\begin{aligned} x_3 &= x_2 - \gamma \nabla f_{e_2}(x_2) \\ &= x_1 - \gamma(\nabla f_{e_2}(x_2) + \nabla f_{e_1}(x_1)) \end{aligned}$$

- The computation of $\nabla f_{e_1}(x_1)$ and $\nabla f_{e_2}(x_2)$ is assigned to CPU1 and CPU2 respectively.

Issues of the Asynchronous Updates



time	x in ∇f		x in mem	∇f in cache		what	
t_1	x_1	x_1	x_1	—	—	read x_1 from memory	
t_2	x_2	x_1	$x_1 - \gamma \nabla f_{e_1}(x_1)$	$\nabla f_{e_1}(x_1)$	—	Update mem.	Computing
t_3	x_2	x_2	$x_2 - \gamma \nabla f_{e_2}(x_1)$	$\nabla f_{e_1}(x_1)$	$\nabla f_{e_2}(x_1)$	Idle	Update mem.

- What we should get: $x_1 - \gamma(\nabla f_{e_1}(x_1) + \nabla f_{e_2}(x_2))$.
- What we actually get: $x_1 - \gamma(\nabla f_{e_1}(x_1) + \nabla f_{e_2}(x_1))$.

Analysis of Hogwild!

Assumptions

- ∇f is Lipschitz continuous.

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|$$

- f is strongly convex with modulus c. Each f_e is convex.

$$f(y) \geqslant f(x) + (y - x)^T \nabla f(x) + \frac{c}{2}\|y - x\|^2$$

- The gradient of f_e is bounded. Recall that $G_e(x) = |E|\nabla f_e(x)$.

$$\|G_e(x)\| \leqslant M, \ \forall \ x$$

- $\gamma c < 1$, otherwise even gradient descent alg will fail.
- Based on the Hogwild!(Variant 1) algorithm.

Proposition 5.1 (Main result).

Suppose the lag between when a gradient is computed and when it is used in step j – namely $j - k(j)$ – is always less or equal than τ, and γ is defined to be

$$\gamma = \frac{\theta \varepsilon c}{2LM^2\Omega(1 + 6\rho\tau + 4\tau^2\Omega\Delta^{1/2})} \tag{2}$$

for some $\varepsilon > 0$ and $\theta \in (0, 1)$. Define $D_0 = \|x_0 - x_\star\|^2$ and let k be an integer satisfying

$$k \geqslant \frac{2LM^2\Omega(1 + 6\tau\rho + 6\tau^2\Omega\Delta^{1/2}) \log(LD_0/\varepsilon)}{c^2\theta\varepsilon} \tag{3}$$

Then after k component updates of x, we have $\mathbb{E}[f(x_k) - f_\star] \leqslant \varepsilon$.

Remarks on Prop 5.1

- Consider the Hogwild! algorithm as an SGD with lags.

$$x_{j+1} \leftarrow x_j - \gamma |e| \mathcal{P}_v G_e(x_{k(j)})$$

- The lags should be bounded by τ. That is, at the jth update, the point used to compute the gradient is within the last τ steps.
- τ is proportional to the number of threads.
- The step size γ should be $\mathcal{O}(\varepsilon)$ to ensure the convergence.
- To obtain an accuracy of ε, we need at least $\mathcal{O}(1/\varepsilon)$ updates, modulo the $\log(1/\varepsilon)$ factor.

Outline

Problem Definition

Consider

$$\min_{x \in \mathbb{R}^d} f(x) = \sum_{e \in E} f_e(x_e)$$

- e denotes a small subset of $\{1, 2, \ldots, d\}$.
- In many of machine learning problems, d and $n = |E|$ are both large.
- f_e only acts on a few components of x, say x_e.
- f_e can easily be regarded as a function over $\mathbb{R}^d$, just by ignoring the components not in the subset e.

Algorithms to Solve the Problem

- HOGWILD! (Asynchronous SGD):

$$x_{k+1} \leftarrow x_k - \gamma_k \nabla f_{s_k}(x_k)$$

- All cores perform the updates asynchronously with no memory locking.
- Ignores the conflict variables.
- No convergence in some cases. (Lose some of the properties of SGD)
- Complicated theoretical analysis |>_<|

Another Way?

To deal with conflicts:

- Traditional parallelizations
  - Lock the conflict variables when updating.
  - Ensure the correctness, but only limited speedup.
- HOGWILD!
  - Ignore the conflicts when updating.
  - Better speedup, but higher risk of not converging.
- CYCLADES
  - Avoid the conflicts by rearranging the updates.
  - Better speedup.
  - Preserve the property of SGD in high probability.

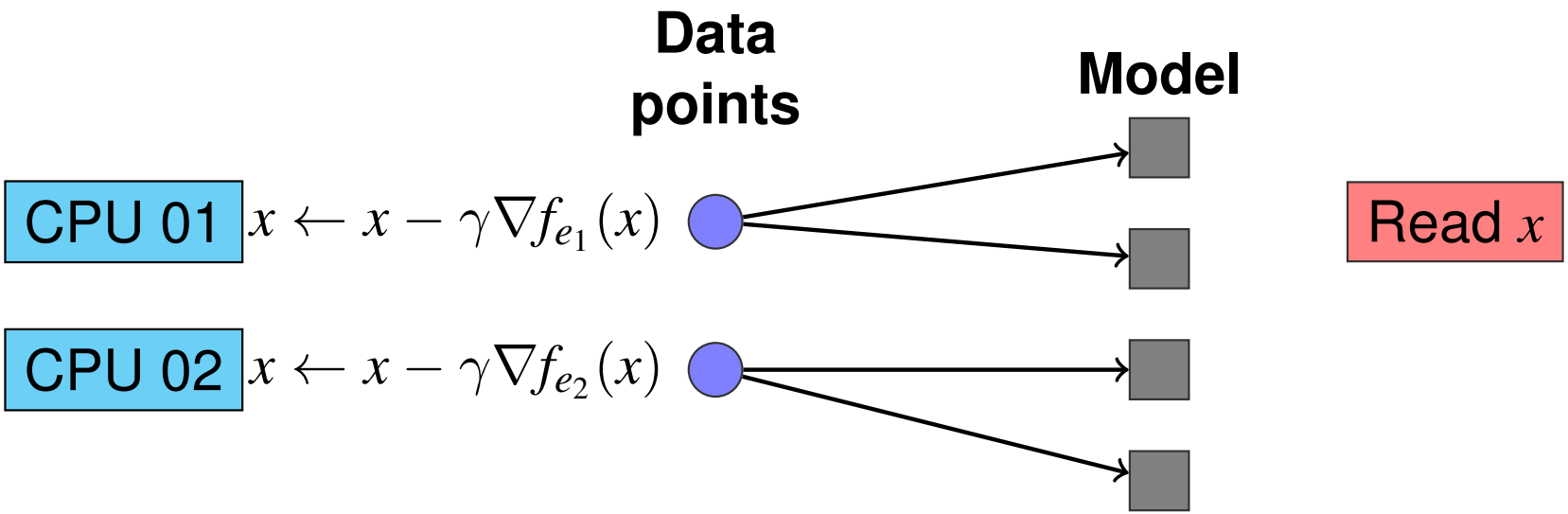
About the Conflicts

Why conflicts affect so much?

- CPUs don't have direct access to the memory.
- Computations can only be done on CPU cache.
- Data are read from the memory to CPU cache. After the computations are finished, results are pushed back to the memory.
- A CPU has no idea whether its 'colleagues' have local updates which have not been pushed to the memory.

About the Conflicts

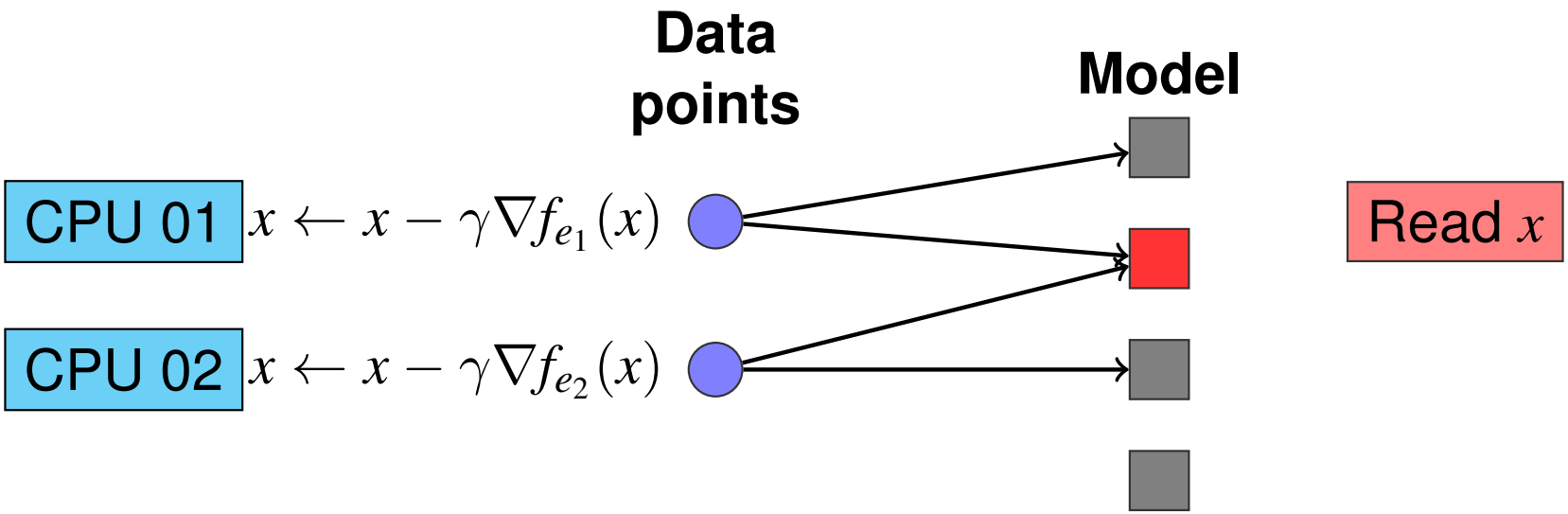
Good Case:



- The components of f_{e_1} and f_{e_2} don't overlap.
- Two parallel updates means two serial updates.
- No conflicts means speed up!

About the Conflicts

Bad Case:



- The components of f_{e_1} and f_{e_2} overlap at x_2.
- Conflicts mean less parallelization.
- What should be done to handle the conflicts? HOGWILD! tells us to ignore it. CYCLADES tells us to avoid it.

The Updates Conflict Graph

Definition 1 (Bipartite update-variable graph).

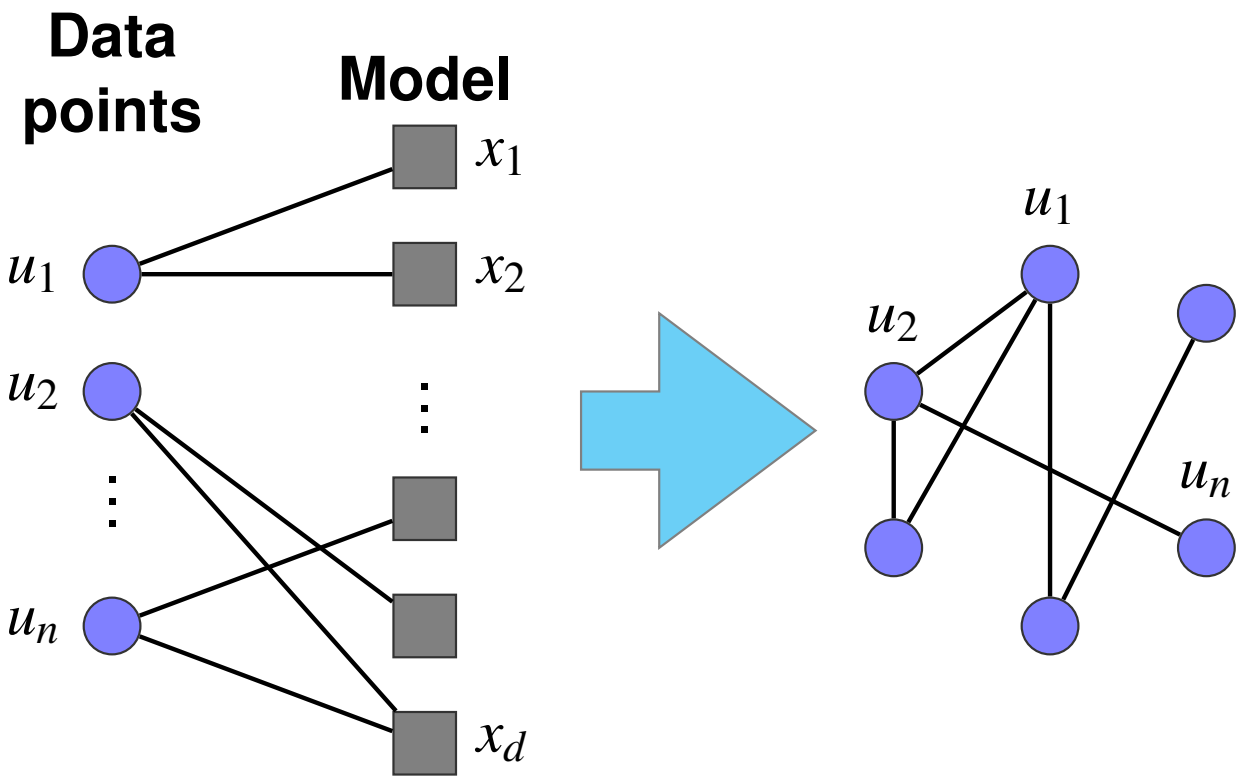
We denote as G_u the bipartite update-variable graph between the updates $u_1, \ldots, u_n$ and the d variables. In G_u an update u_i is linked to a variable x_j, if u_i requires to read and write x_j. E_u denotes the number of edges in the bipartite graph. Δ_L denotes the left max vertex degree of G_u. $\bar{\Delta}_L$ denotes its average left degree.

Definition 2 (Conflict Graph).

We denot by G_c a conflict graph on n vertices, each corresponding to an update u_i. Two vertices of G_c are linked with an edge, if and only if the corresponding updates share at least one variable in the bipartite-update graph G_u. Δ is denoted as the max vertex degree of G_c.

The Updates Conflict Graph

From a bipartite graph to a conflict graph



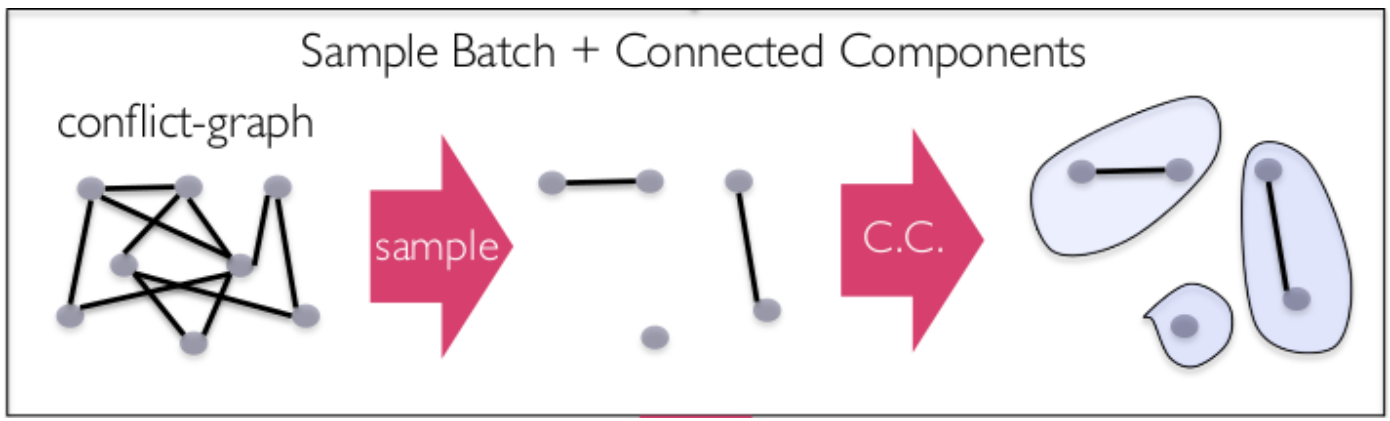
The Updates Conflict Graph

Remarks

- In the conflict graph, if two vertices are linked with an edge, then their corresponding updates conflict with each other.
- G_c is introduced to analyze the CYCLADES algorithm. In practice we never construct it.
- Suppose the number of connected components(CC) is N_{cc}, then we can employ N_{cc} workers to process the updates asynchronously. No conflicts occur. **However,** N_{cc} **equals to 1 for most problems!**

The Idea of CYCLADES

- We need more CCs to perform parallelization.
- Just sample part of the updates.



How many CCs we can get if we perform the sampling?

The Number of CCs

Theorem 2.

Let G be a graph on n vertices, with maximum vertex degree Δ. Let us sample each vertex independently with probability $p = \frac{1-\varepsilon}{\Delta}$ and define G' as the induced subgraph on sampled vertices. Then, the largest CC of G' has the size at most $\frac{4}{\varepsilon^2} \log n$, with high probability.

Theorem 3.

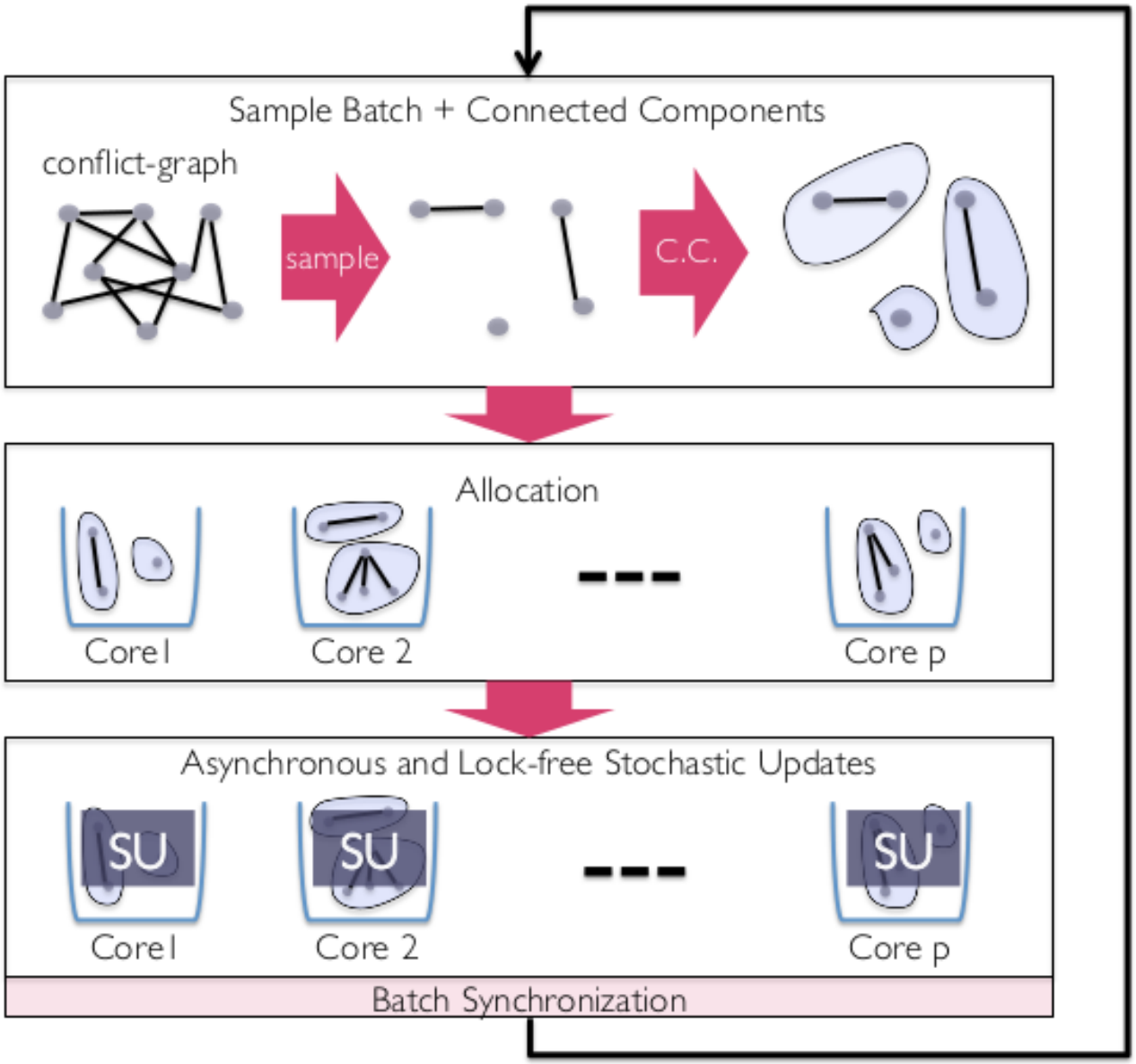
Let G be a graph on n vertices, with maximum vertex degree Δ. Let us sample $B = (1-\varepsilon)\frac{n}{\Delta}$, with or without replacement, and define G' as the induced subgraph on sampled vertices. Then, the largest CC of G' has the size at most $\mathcal{O}\left(\frac{\log n}{\varepsilon^2}\right)$, with high probability.

The Number of CCs

Remarks

- Theorem 3 can be regard as a corollary of Theorem 2(Not obvious!). We'll give a brief proof of Theorem 2 later.
- From Theorem 3, if one samples $B = (1-\varepsilon)\frac{n}{\Delta}$ vertices, then there will be at least $\mathcal{O}(\varepsilon^2 B/\log n)$ CCs, each of size at most $\mathcal{O}(\log n/\varepsilon^2)$.
- The number of CCs is increased a lot – OK. Good for parallelization.

The Idea of CYCLADES



The Idea of CYCLADES

- After sampling, use certain algorithm to find all CCs in a parallel way(to be continued).
- When all CCs are determined, allocate them to the workers(consider the load balance).
- Each worker performs stochastic updates asynchronously and individually, with no conflicts and false sharing issues.
- Repeat the three phases until finished.

CYCLADES Outline

Algorithm 3 CYCLADES

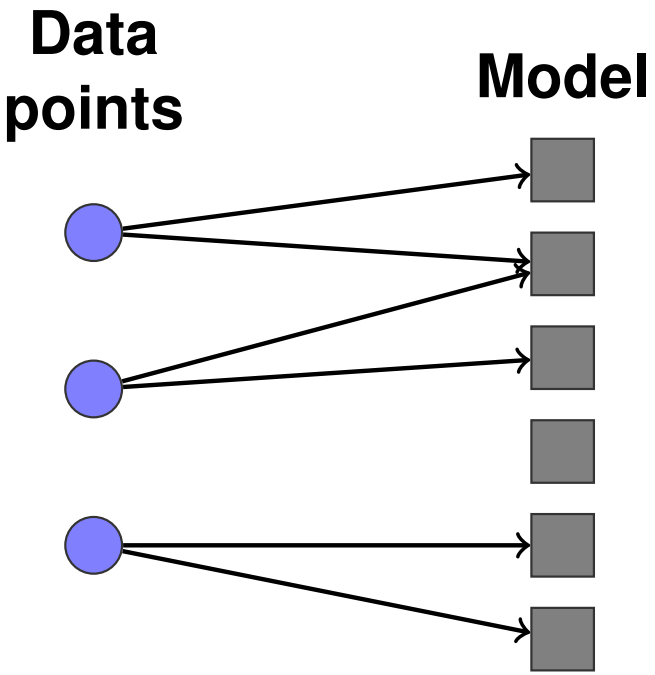
1: **Input**: G_u, T, B.
2: Sample $n_b = T/B$ subgraphs from G_u.
3: **for** batch $i = 1 : n_b$ **do**
4: Allocate CCs to P cores.
5: **for** each core asynchronously **do**
6: Perform local updates in allocated CCs.
7: **end for**
8: **end for**

- T is the total updates you'd like to run.
- One synchronization at the end of each batch.
- G_u is determined by the structure of problem.

Compute CCs of All Batches in Parallel

- If we use the conflict graph G_c to compute the CCs...
- The cost is the order of sampled edges.
- However, constructing G_c requires n^2 time. A high cost – no, thanks.
- Compute CCs using the bipartite graph only!

Compute CCs of All Batches in Parallel



Simple message passing idea

- Data points send their label.
- Variables compute min and send back.
- Data points compute min.
- Iterate until you're done.

Compute CCs of All Batches in Parallel

Remarks

- The iterations needed are bounded with the length of the longest-shortest path(i.e. the diameter of G_c).
- Overall complexity: $\mathcal{O}(E_u \log n/P)$, P is the number of cores. Note: useful when P is large.
- Only need the bipartite graph G_u, not the conflict graph G_c.

Allocating the Conflict Groups

Once we obtain the CCs of the batch, we have to allocate these CCs to P cores.

- w_i is the cost of updating with the ith data point.
- $W_{C(i)} = \sum_{j \in C(i)} w_j$ is the cost of updating the ith CC.
- For each batch update, we need to

$$\min \max W_{C(i)}$$

  to get the best load balance.
- An NP hard problem. We can utilize an approximation algorithm to obtain a sub-optimal plan.

Allocating the Conflict Groups

Algorithm 4 Greedy allocation

1: Estimate $W_1, \ldots, W_m$, the cost of each CC.
2: Sort the W_i's in the descending order.
3: **for** $i = 1 : m$ **do**
4: Choose the currently largest W_i.
5: Add W_i to the cores with least sum of cost currently.
6: **end for**

- A $4/3$-approximation algorithm.
- w_i is proportional to the out-degree of that update.

Main Results

Theorem 4 (Main result).

Let us assume any given update-variable graph G_u with average, and max left degree $\bar{\Delta}_L$ and Δ_L, such that $\Delta_L/\bar{\Delta}_L \leqslant \sqrt{n}$, and with induced max conflict degree Δ. Then, CYCLADES on $P = \mathcal{O}(n/\Delta\Delta_L)$ cores with batch sizes $B = (1-\varepsilon)\frac{n}{\Delta}$ can execute $T = cn$ updates, for any $c > 1$, in time

$$\mathcal{O}(\frac{E_u \cdot \kappa}{P} \log^2 n)$$

with high probability.

- κ is the cost to update one edge in E_u.
- CYCLADES can achieve the same result as the serial algorithms with high probability. Thus it requires similar properties of the objective function f and its components f_e.

Main Results

Remarks of Thm 4

- The sampling of each batch can be with or without replacement.
- The number of cores is bounded, for it's hard to allocate the work of computing CCs evenly if P is too large.
- The batch size B should be bounded. This is very important. Small B will induce a subgraph with many CCs.
- Theorem 2 is the most important foundation to develop the main result.