分块坐标下降法

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

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

问题形式

考虑具有如下形式的问题：

$$\min_{x\in\mathcal{X}} \quad F(x_1,x_2,\cdots,x_s)=f(x_1,x_2,\cdots,x_s)+\sum_{i=1}^{s} r_i(x_i), \tag{1}$$

- $\mathcal{X}$是函数的可行域，自变量x拆分成s个变量块$x_1,x_2,\cdots,x_s$，每个变量块$x_i\in\mathbb{R}^{n_i}$.
- 函数f是关于x的可微函数，每个$r_i(x_i)$关于x_i是适当的闭凸函数，但不一定可微.
- 目标函数F的性质体现在f，每个r_i以及自变量的分块上. 通常情况下，f对于所有变量块x_i不可分，但单独考虑每一块自变量时，f有简单结构；r_i只和第i个自变量块有关，因此r_i在目标函数中是一个可分项.
- 求解问题(1) 的难点在于如何利用分块结构处理不可分的函数f.

问题形式

- 分组LASSO 模型：参数$x=(x_1,x_2,\cdots,x_G)\in\mathbb{R}^p$ 可以分成G 组，且$\{x_i\}_{i=1}^G$中只有少数的非零向量.

$$\min_x \quad \frac{1}{2n}\|b-Ax\|_2^2+\lambda\sum_{i=1}^{G}\sqrt{p_i}\|x_i\|_2.$$

- K-均值聚类问题的等价形式：

$$\begin{aligned}\min_{\Phi,H}\quad & \|A-\Phi H\|_F^2,\\ \text{s.t.}\quad & \Phi\in\mathbb{R}^{n\times k},\text{每一行只有一个元素为1，其余为0},\\ & H\in\mathbb{R}^{k\times p}.\end{aligned}\tag{2}$$

- 低秩矩阵恢复：设$b\in\mathbb{R}^m$是已知的观测向量，$\mathcal{A}$是线性映射.

$$\min_{X,Y}\quad \frac{1}{2}\|\mathcal{A}(XY)-b\|_2^2+\alpha\|X\|_F^2+\beta\|Y\|_F^2,$$

其中$\alpha,\beta>0$为正则化参数.

问题形式

- 非负矩阵分解：设$\mathcal{M}$是已知张量，考虑求解如下极小化问题：

$$\min_{A_1,A_2,\cdots,A_N\geq 0} \quad \frac{1}{2}\|\mathcal{M}-A_1\circ A_2\circ\cdots\circ A_N\|_F^2+\sum_{i=1}^{N}\lambda_i r_i(A_i),$$

其中“$\circ$”表示张量的外积运算.

- 字典学习：设$A\in\mathbb{R}^{m\times n}$为$n$个观测，每个观测的信号维数是$m$，现在我们要从$A$中学习出一个字典$D\in\mathbb{R}^{m\times k}$和系数矩阵$X\in\mathbb{R}^{k\times n}$：

$$\begin{aligned}\min_{D,X} \quad & \frac{1}{2n}\|DX-A\|_F^2+\lambda\|X\|_1,\\ \text{s.t.} \quad & \|D\|_F\leq 1.\end{aligned}$$

在这里自变量有两块，分别为D和X，此外对D还存在球约束$\|D\|_F\leq 1$.

挑战和难点

- Non-convexity and non-smoothness cause trouble
- tricky convergence analysis;
- expensive updates to all variables simultaneously.
- **Goal**: to develop an efficient algorithm with simple update and global convergence (of course, to a stationary point

变量划分

- **分块坐标下降法更新方式**：按照$x_1, x_2, \cdots, x_s$的次序依次固定其他$(s-1)$块变量极小化F，完成一块变量的极小化后，它的值便立即被更新到变量空间中，更新下一块变量时将使用每个变量最新的值.

- 变量划分

$$\mathcal{X}_i^k = \{x \in \mathbb{R}^{n_i} \mid (x_1^k, \cdots, x_{i-1}^k, x, x_{i+1}^{k-1}, \cdots, x_s^{k-1}) \in \mathcal{X}\}.$$

- 辅助函数

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

其中x_j^k表示在第k次迭代中第j块自变量的值，函数f_i^k表示在第k次迭代更新第i块变量时所需要考虑的目标函数的光滑部分.

变量更新方式

在每一步更新中，通常使用以下三种更新格式之一：

$$x_i^k = \underset{x_i \in \mathcal{X}_i^k}{\operatorname{argmin}} \left\{ f_i^k(x_i) + r_i(x_i) \right\}, \tag{3}$$

$$x_i^k = \underset{x_i \in \mathcal{X}_i^k}{\operatorname{argmin}} \left\{ f_i^k(x_i) + \frac{L_i^{k-1}}{2} \|x_i - x_i^{k-1}\|_2^2 + r_i(x_i) \right\}, \tag{4}$$

$$x_i^k = \underset{x_i \in \mathcal{X}_i^k}{\operatorname{argmin}} \left\{ \langle \hat{g}_i^k, x_i - \hat{x}_i^{k-1} \rangle + \frac{L_i^{k-1}}{2} \|x_i - \hat{x}_i^{k-1}\|_2^2 + r_i(x_i) \right\}, \tag{5}$$

- $L_i^k > 0$为常数。
- 在更新格式(5)中，$\hat{x}_i^{k-1}$采用外推定义：

$$\hat{x}_i^{k-1} = x_i^{k-1} + \omega_i^{k-1}(x_i^{k-1} - x_i^{k-2}), \tag{6}$$

其中$\omega_i^k \geq 0$为外推的**权重**，$\hat{g}_i^k \overset{\text{def}}{=\!=} \nabla f_i^k(\hat{x}_i^{k-1})$为外推点处的梯度.

算法格式

Block Coordinate Descent (BCD) Method

Algorithm 1 分块坐标下降法

1: **初始化**：选择两组初始点$(x_1^{-1}, x_2^{-1}, \cdots, x_s^{-1}) = (x_1^0, x_2^0, \cdots, x_s^0)$.
2: **for** $k = 1, 2, \cdots$ **do**
3: **for** $i = 1, 2, \cdots$ **do**
4: 使用格式(3) 或(4) 或(5) 更新x_i^k.
5: **end for**
6: **if** 满足停机条件 **then**
7: 返回$(x_1^k, x_2^k, \cdots, x_s^k)$，算法终止.
8: **end if**
9: **end for**

- 三种格式都有其适用的问题，特别是子问题是否可写出显式解
- 在每一步更新中，三种迭代格式对不同自变量块可以混合使用，不必仅仅局限于一种.

算法格式

- BCD算法的子问题可采用三种不同的更新格式，这三种格式可能会产生不同的迭代序列，可能会收敛到不同的解，坐标下降算法的数值表现也不相同.
- 格式(3)是最直接的更新方式，它严格保证了整个迭代过程的目标函数值是下降的. 然而由于f的形式复杂，子问题求解难度较大. 在收敛性方面，格式(3)在强凸问题上可保证目标函数收敛到极小值，但在非凸问题上不一定收敛.
- 格式(4) (5) 则是对格式(3)的修正，不保证迭代过程目标函数的单调性，但可以改善收敛性结果. 使用格式(4)可使得算法收敛性在函数F为非严格凸时有所改善.
- 格式(5)实质上为目标函数的一阶泰勒展开近似，在一些测试问题上有更好的表现，可能的原因是使用一阶近似可以避开一些局部极小值点. 此外，格式(5)的计算量很小，比较容易实现.

例子：二元二次函数

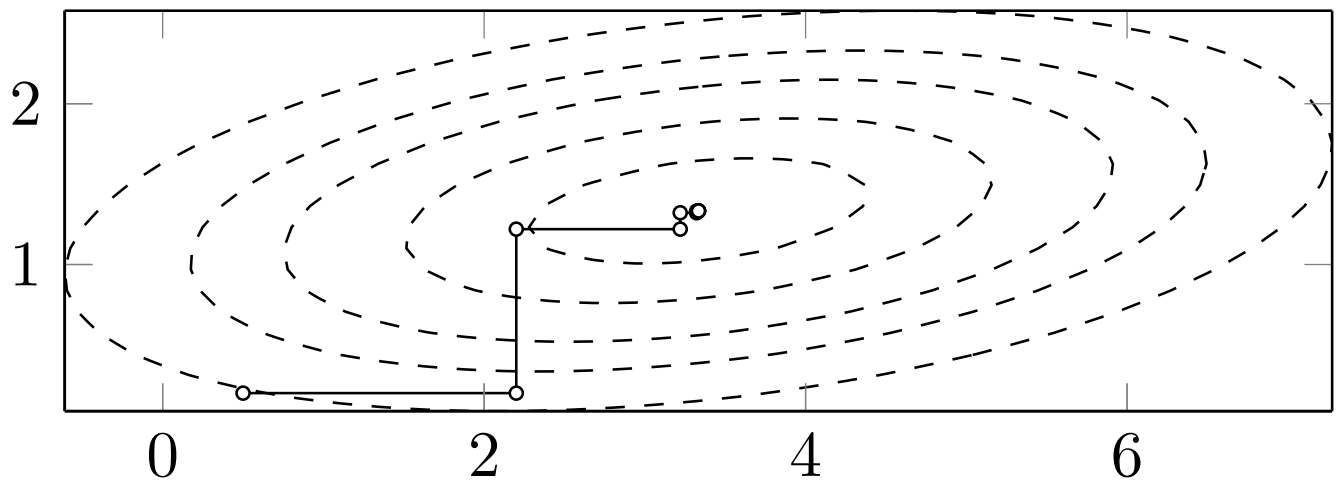
考虑二元二次函数的优化问题

$$\min \quad f(x,y) = x^2 - 2xy + 10y^2 - 4x - 20y.$$

故采用格式(3)的分块坐标下降法为

$$x^{k+1} = 2 + y^k, \quad y^{k+1} = 1 + \frac{x^{k+1}}{10}. \tag{7}$$

下图描绘了当初始点为 $(x,y) = (0.5, 0.2)$时的迭代点轨迹，可以看到在进行了7 次迭代后迭代点与最优解已充分接近.



不收敛反例

值得注意的是，对于非凸函数$f(x)$，分块坐标下降法可能失效．Powell在1973 年就给出了一个使用格式(3)但不收敛的例子:

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

其中$(x_i-1)_+^2$的含义为先对(x_i-1)取正部再平方．设$\varepsilon>0$，初始点取为

$$x^0=\left(-1-\varepsilon,1+\frac{\varepsilon}{2},-1-\frac{\varepsilon}{4}\right),$$

容易验证迭代序列满足

$$x^k=(-1)^k\cdot(-1,1,-1)+(-\frac{1}{8})^k\cdot\left(-\varepsilon,\frac{\varepsilon}{2},-\frac{\varepsilon}{4}\right),$$

这个迭代序列有两个聚点$(-1,1,-1)$与$(1,-1,1)$，但这两个点都不是F的稳定点．

Outline

LASSO 问题求解

下面介绍如何使用分块坐标下降法来求解LASSO 问题

$$\min_x \mu\|x\|_1 + \frac{1}{2}\|Ax-b\|^2. \tag{8}$$

将自变量x记为$x=\left[x_i, \bar{x}_i^\top\right]^\top$，其中$\bar{x}_i$为$x$去掉第$i$个分量而形成的列向量．而相应地，矩阵$A$在第$i$块的更新记为$A=\begin{bmatrix} a_i & \bar{A}_i \end{bmatrix}$，其中$\bar{A}_i$为矩阵$A$去掉第$i$列而形成的矩阵．

在第i块的更新中考虑格式(3)。做替换$c_i = b - \bar{A}_i\bar{x}_i$，原问题等价于

$$\min_{x_i} \quad f_i(x_i) \overset{\text{def}}{=\!=} \mu|x_i| + \frac{1}{2}\|a_i\|^2 x_i^2 - a_i^{\mathrm{T}} c_i x_i. \tag{9}$$

可直接写出它的最小值点

$$x_i^k = \operatorname*{argmin}_{x_i} f_i(x_i) = \begin{cases} \frac{a_i^{\mathrm{T}}c_i-\mu_i}{\|a_i\|^2}, & a_i^{\mathrm{T}}c_i > \mu, \\ \frac{a_i^{\mathrm{T}}c_i+\mu_i}{\|a_i\|^2}, & a_i^{\mathrm{T}}c_i < -\mu, \\ 0, & \text{其他}. \end{cases} \tag{10}$$

K-均值聚类算法

- 当固定H时，设Φ的每一行为ϕ_i^{T}，那么根据矩阵分块乘法，

$$A-\Phi H=\begin{bmatrix}a_1^{\mathrm{T}}\\ a_2^{\mathrm{T}}\\ \vdots\\ a_n^{\mathrm{T}}\end{bmatrix}-\begin{bmatrix}\phi_1^{\mathrm{T}}\\ \phi_2^{\mathrm{T}}\\ \vdots\\ \phi_n^{\mathrm{T}}\end{bmatrix}H=\begin{bmatrix}a_1^{\mathrm{T}}-\phi_1^{\mathrm{T}}H\\ a_2^{\mathrm{T}}-\phi_2^{\mathrm{T}}H\\ \vdots\\ a_n^{\mathrm{T}}-\phi_n^{\mathrm{T}}H\end{bmatrix}.$$

注意到ϕ_i只有一个分量为1，其余分量为0，不妨设其第j个分量为1，此时$\phi_i^{\mathrm{T}}H$ 相当于将H 的第j行取出，因此$\|a_i^{\mathrm{T}}-\phi_i^{\mathrm{T}}H\|$为$a_i^{\mathrm{T}}$与$H$的第$j$个行向量的距离．我们的最终目的是极小化$\|A-\Phi H\|_F^2$，所以$j$应该选矩阵$H$中距离$a_i^{\mathrm{T}}$最近的那一行，即

$$\Phi_{ij}=\begin{cases}1, & j=\underset{l}{\operatorname{argmin}}\,\|a_i-h_l\|,\\ 0, & \text{其他}.\end{cases}$$

其中h_l^{T}表示矩阵H的第l行．

K-均值聚类算法

- 当固定Φ时，此时考虑H的每一行h_j^{T}，根据目标函数的等价性有

$$\|A-\Phi H\|_F^2=\sum_{j=1}^{k}\sum_{a\in S_j}\|a-h_j\|^2,$$

因此只需要对每个h_j求最小即可．设$\bar{a}_j$是目前第j类所有点的均值，则

$$\begin{aligned}\sum_{a\in S_j}\|a-h_j\|^2&=\sum_{a\in S_j}\|a-\bar{a}_j+\bar{a}_j-h_j\|^2\\&=\sum_{a\in S_j}\left(\|a-\bar{a}_j\|^2+\|\bar{a}_j-h_j\|^2+2\left\langle a-\bar{a}_j,\bar{a}_j-h_j\right\rangle\right)\\&=\sum_{a\in S_j}\left(\|a-\bar{a}_j\|^2+\|\bar{a}_j-h_j\|^2\right),\end{aligned}$$

这里利用了交叉项$\sum_{a\in S_j}\left\langle a-\bar{a}_j,\bar{a}_j-h_j\right\rangle=0$的事实．因此容易看出，此时$h_j$直接取为$\bar{a}_j$即可达到最小值

非负矩阵分解

考虑最基本的非负矩阵分解问题

$$\min_{X,Y\geq 0} \quad f(X,Y)=\frac{1}{2}\|XY-M\|_F^2. \tag{11}$$

可以计算梯度

$$\frac{\partial f}{\partial X}=(XY-M)Y^{\mathrm{T}},\quad \frac{\partial f}{\partial Y}=X^{\mathrm{T}}(XY-M). \tag{12}$$

注意到在格式(5)中，当$r_i(X)$为凸集示性函数时即是求解到该集合的投影，因此得到分块坐标下降法如下：

$$\begin{aligned} X^{k+1}&=\max\{X^k-t_k^x(X^kY^k-M)(Y^k)^{\mathrm{T}},0\},\\ Y^{k+1}&=\max\{Y^k-t_k^y(X^k)^{\mathrm{T}}(X^kY^k-M),0\}, \end{aligned} \tag{13}$$

其中t_k^x,t_k^y是步长，

$$\min_{D,X} \quad \frac{1}{2n}\|DX - A\|_F^2 + \lambda\|X\|_1 + \frac{\mu}{2}\|D\|_F^2. \tag{14}$$

- 当固定变量D时，考虑函数

$$f_D(X) = \frac{1}{2n}\|DX - A\|_F^2 + \lambda\|X\|_1.$$

使用格式(5)．通过直接计算可得$f_D(X)$中光滑部分的梯度为

$$G = \frac{1}{n}D^{\mathrm{T}}(DX - A),$$

因此格式(5)等价于

$$X^{k+1} = \mathrm{prox}_{t_k\lambda\|\cdot\|_1}\left(X^k - \frac{t_k}{n}(D^k)^{\mathrm{T}}(D^kX^k - A)\right),$$

其中t_k为步长．

$$\min_{D,X} \quad \frac{1}{2n}\|DX - A\|_F^2 + \lambda\|X\|_1 + \frac{\mu}{2}\|D\|_F^2. \tag{15}$$

- 当固定变量X时，考虑函数

$$f_X(D) = \frac{1}{2n}\|DX - A\|_F^2 + \frac{\mu}{2}\|D\|_F^2.$$

使用格式(3)．计算关于D^{T}的梯度为

$$\nabla_{D^{\mathrm{T}}} f_X(D) = \frac{1}{n}X(X^{\mathrm{T}}D^{\mathrm{T}} - A^{\mathrm{T}}) + \mu D^{\mathrm{T}},$$

令梯度为零向量，可得

$$D = AX^{\mathrm{T}}(XX^{\mathrm{T}} + n\mu I)^{-1}.$$

因为$X \in \mathbb{R}^{k\times n}$，其中$k \ll n$，所以$XX^{\mathrm{T}}$是一个比较小的矩阵，可以方便地求出它的逆．故格式(3)等价于

$$D^{k+1} = A(X^{k+1})^{\mathrm{T}}(X^{k+1}(X^{k+1})^{\mathrm{T}} + n\mu I)^{-1}.$$

最大割问题的非凸松弛

最大割问题

$$
\begin{aligned}
&\text{(半定松弛)} \quad \min \quad \langle C, X \rangle, \\
&\qquad\qquad \text{s.t.} \quad X_{ii} = 1,\ i = 1, 2, \cdots, n, \\
&\qquad\qquad\qquad\ \ X \succeq 0. \\
&\text{(非凸松弛)} \quad \min \quad \left\langle C, V^{\mathrm{T}} V \right\rangle, \\
&\qquad\qquad \text{s.t.} \quad v_i \in \mathbb{R}^p,\ \|v_i\| = 1,\ i = 1, 2, \cdots, n, \\
&\qquad\qquad\qquad\ \ V = [v_1, v_2, \cdots, v_n].
\end{aligned}
\tag{16}
$$

- 比较两种松弛方式可知，非凸松弛通过引入分解$X = V^{\mathrm{T}}V$并限制V的每一列的ℓ_2范数为1，将半定松弛中的X对角线元素为1以及X半正定的约束消去了.
- 这两个问题一般不等价，当p充分大时二者等价. 实际计算中通常选取一个较小的p.

最大割问题的非凸松弛

矩阵V是按列分成n块的，考虑格式(3)为例，取定i，固定其余v_j

$$\mathrm{Tr}\left(\begin{bmatrix} C_{11} & \cdots & C_{1i} & \cdots & C_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ C_{i1} & \cdots & C_{ii} & \cdots & C_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ C_{n1} & \cdots & C_{ni} & \cdots & C_{nn} \end{bmatrix} \begin{bmatrix} v_1^{\mathrm{T}}v_1 & \cdots & v_1^{\mathrm{T}}v_i & \cdots & v_1^{\mathrm{T}}v_n \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ v_i^{\mathrm{T}}v_1 & \cdots & v_i^{\mathrm{T}}v_i & \cdots & v_i^{\mathrm{T}}v_n \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ v_n^{\mathrm{T}}v_1 & \cdots & v_n^{\mathrm{T}}v_i & \cdots & v_n^{\mathrm{T}}v_n \end{bmatrix}\right),$$

根据以上矩阵分块示意图可知和v_i有关的部分为

$$C_{ii}v_i^{\mathrm{T}}v_i + \sum_{j\neq i}(C_{ij}+C_{ji})v_i^{\mathrm{T}}v_j.$$

由于约束$\|v_i\| = 1$，上式中第一项是常数。最终在第i步子问题是：

$$\min f_i(v_i) = \left(\sum_{j\neq i} C_{ji}v_j^{\mathrm{T}}\right)v_i, \mathrm{s.t.}\|v_i\| = 1.$$

其解为：$v_i = -\left(\sum_{j\neq i} C_{ji}v_j\right) \Big/ \left\|\sum_{j\neq i} C_{ji}v_j\right\|.$

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 $x \in \operatorname{dom} \sigma$, 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 $\bar{u} \in \mathrm{dom}\ \partial\sigma := \{u \in \mathbb{R}^d \mid \partial\sigma(u) \neq \emptyset\}$: 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 $\mathrm{dom}\ \partial\sigma$, 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 $\mathrm{dom}\sigma$.

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 $u \in \mathrm{dom}\sigma$ 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 $\rho_1 = \min\{(\gamma_1 - 1)\lambda_1^-, (\gamma_2 - 1)\lambda_2^-\}$:

$$\frac{\rho_1}{2}\|z^{k+1} - z^k\|^2 \leq \Psi(z^k) - \Psi(z^{k+1}), \forall k \geq 0 \tag{17}$$

- 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 (17):

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

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

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)

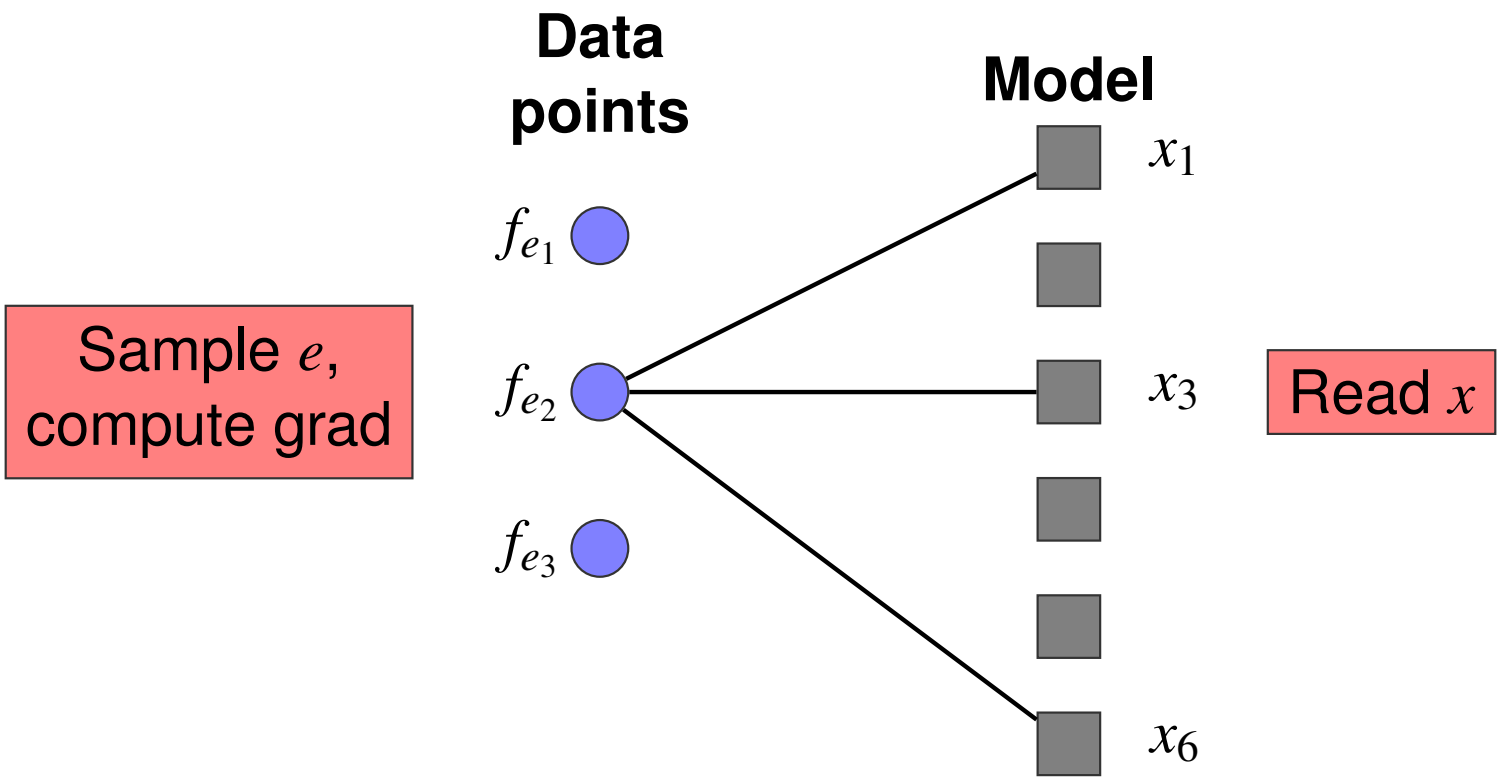
$$x_{k+1} = x_k - \gamma_k \nabla f_{s_k}(x_k)$$

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?

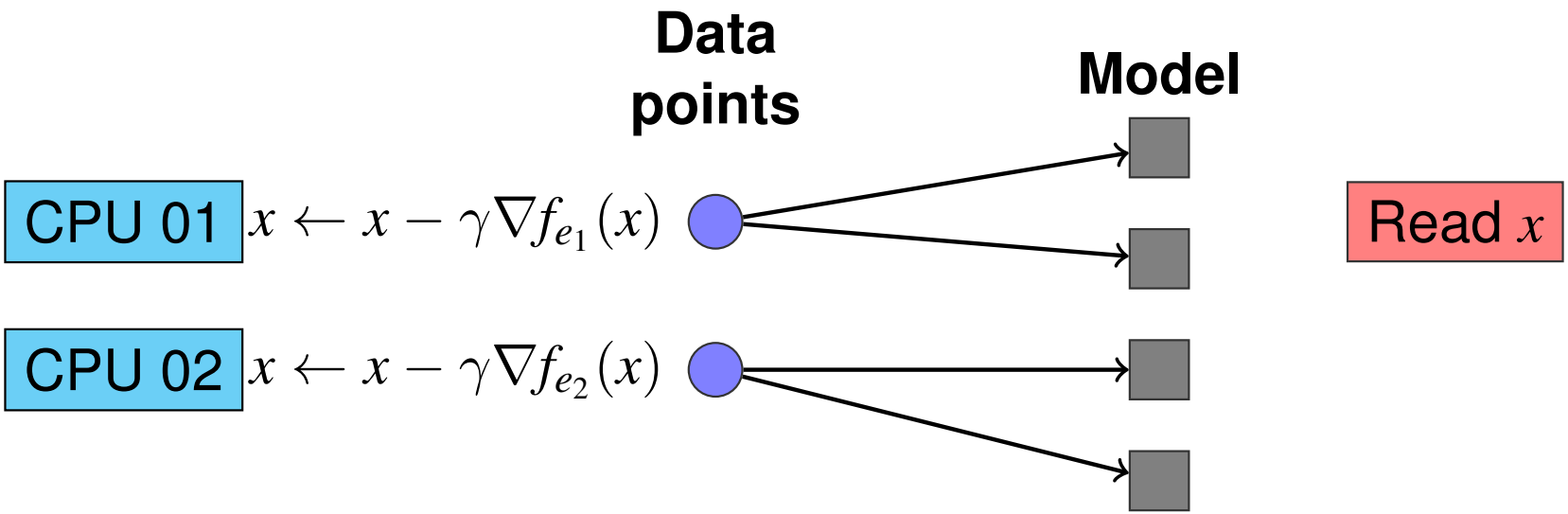
- No – for most cases.
- Almost yes – for problems with structures of sparsity.
- For our problem:

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

  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

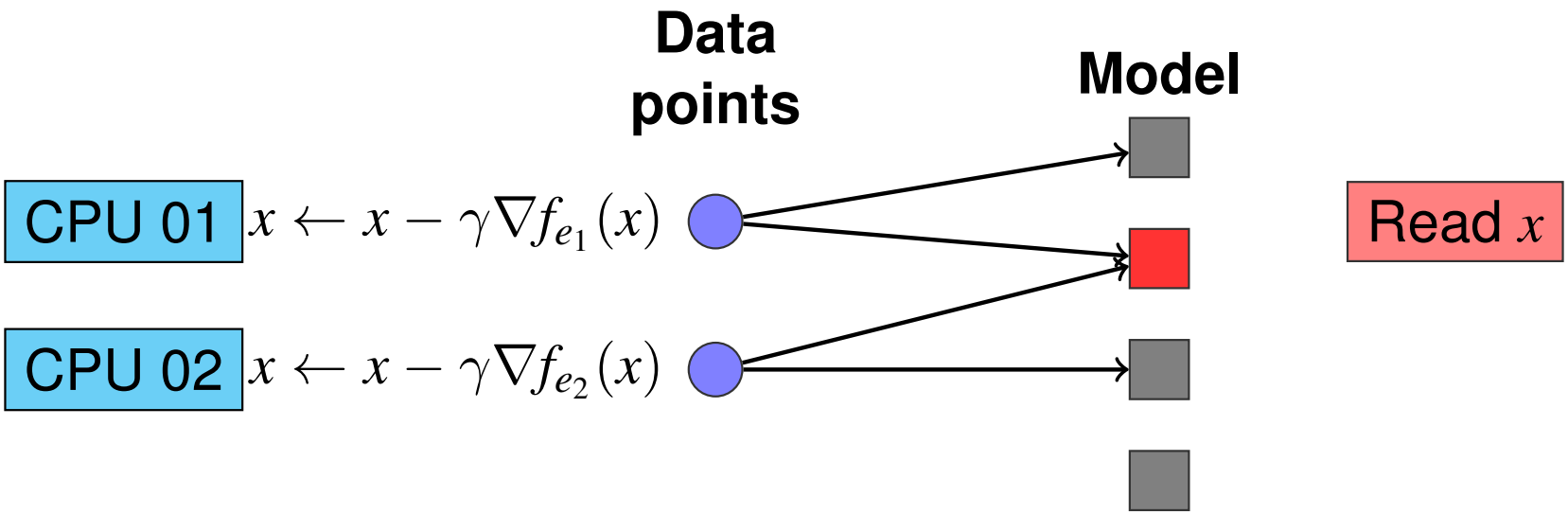
Good Case:



- 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

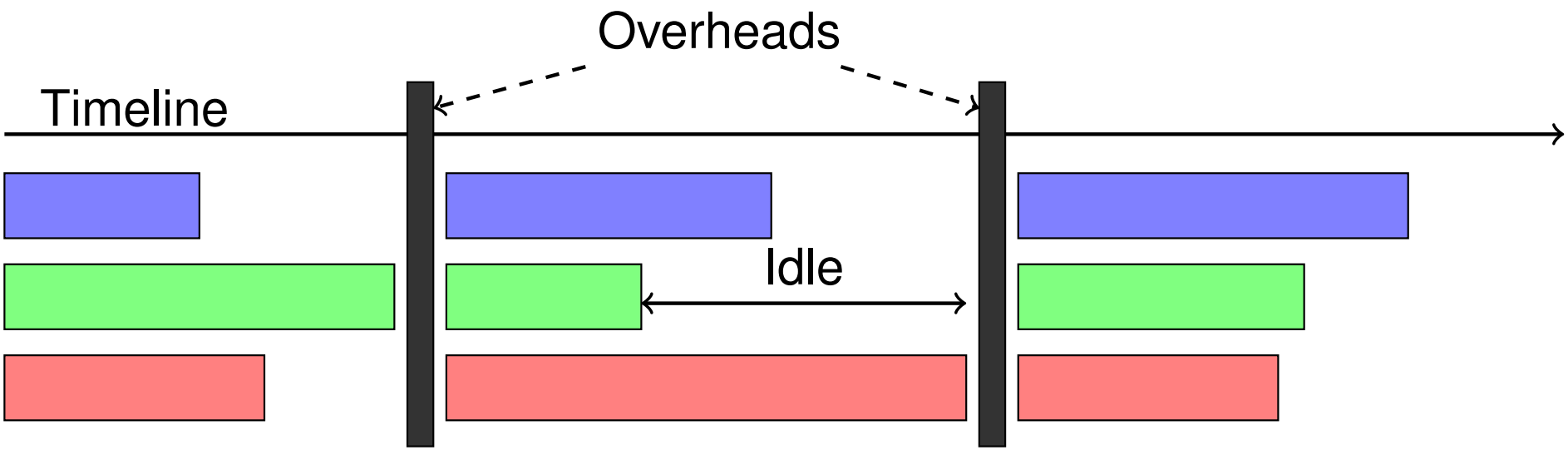
How to deal with the conflicts?

- 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.

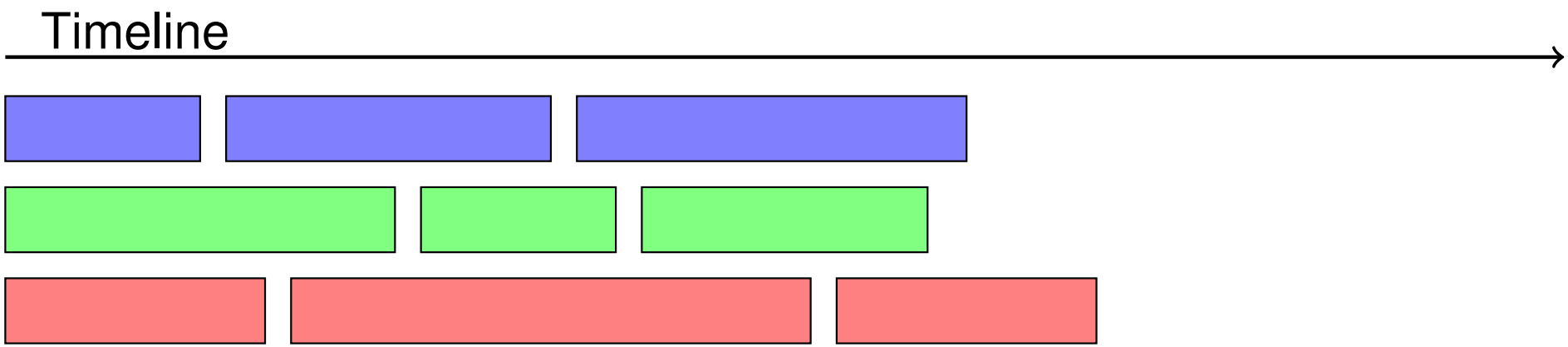
The synchronous world:



- 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 2 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 3 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.

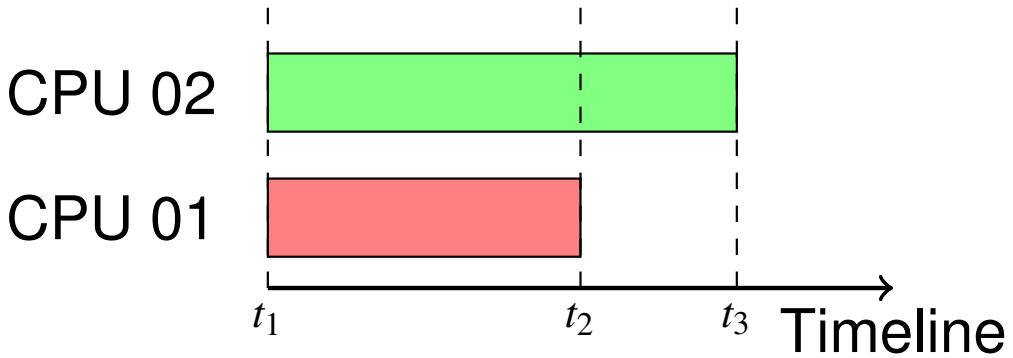
Issues of the Asynchronous Updates

- 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 4.1 (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

$$\gamma = \frac{\theta \varepsilon c}{2LM^2\Omega(1 + 6\rho\tau + 4\tau^2\Omega\Delta^{1/2})} \tag{18}$$

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

$$k \geqslant \frac{2LM^2\Omega(1 + 6\tau\rho + 6\tau^2\Omega\Delta^{1/2}) \log(LD_0/\varepsilon)}{c^2\theta\varepsilon} \tag{19}$$

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

Remarks on Prop 4.1

- 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.

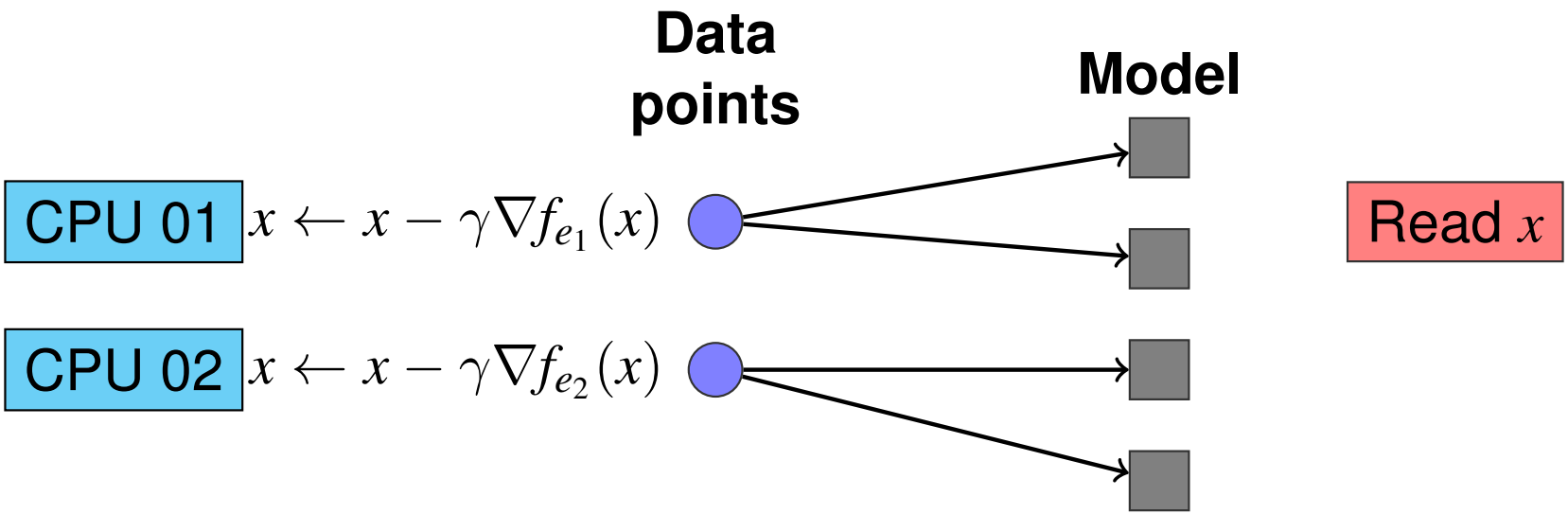
About the Conflicts

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

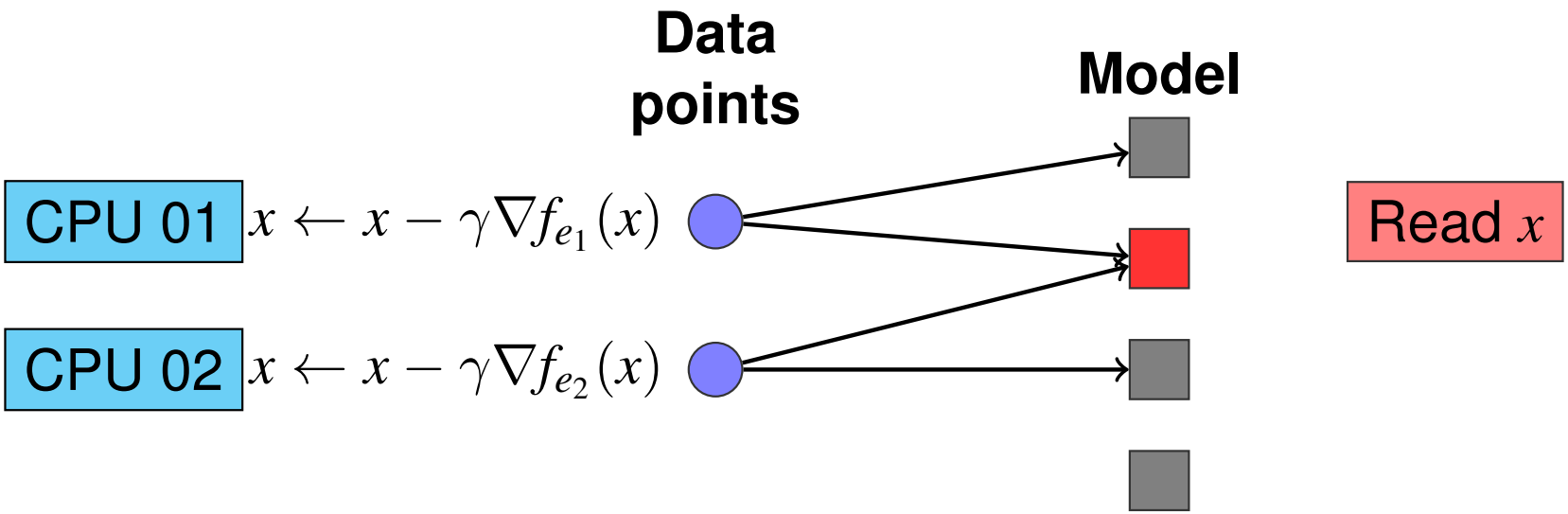
Good Case:



- 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).

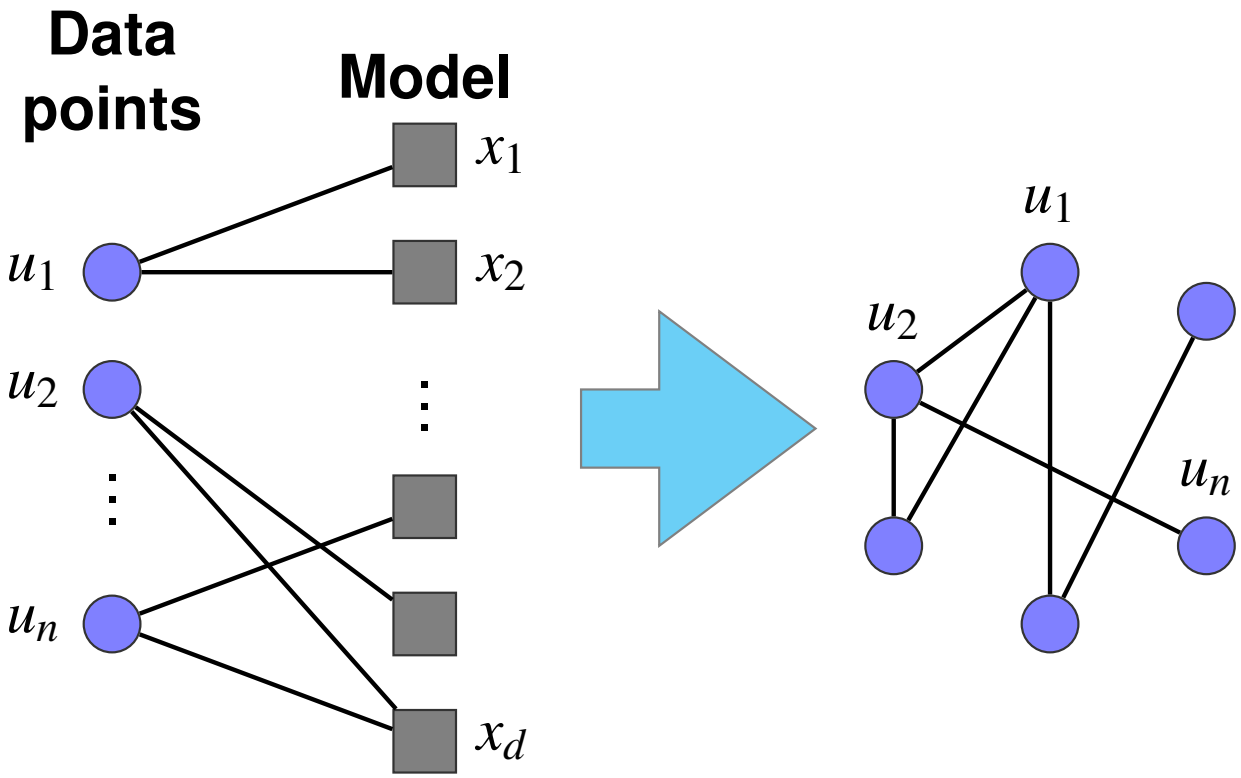
We denote as G_u the bipartite update-variable graph between the updates $u_1, \ldots, u_n$ and the d variables. In G_u an update u_i is linked to a variable x_j, if u_i requires to read and write x_j. E_u denotes the number of edges in the bipartite graph. Δ_L denotes the left max vertex degree of G_u.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



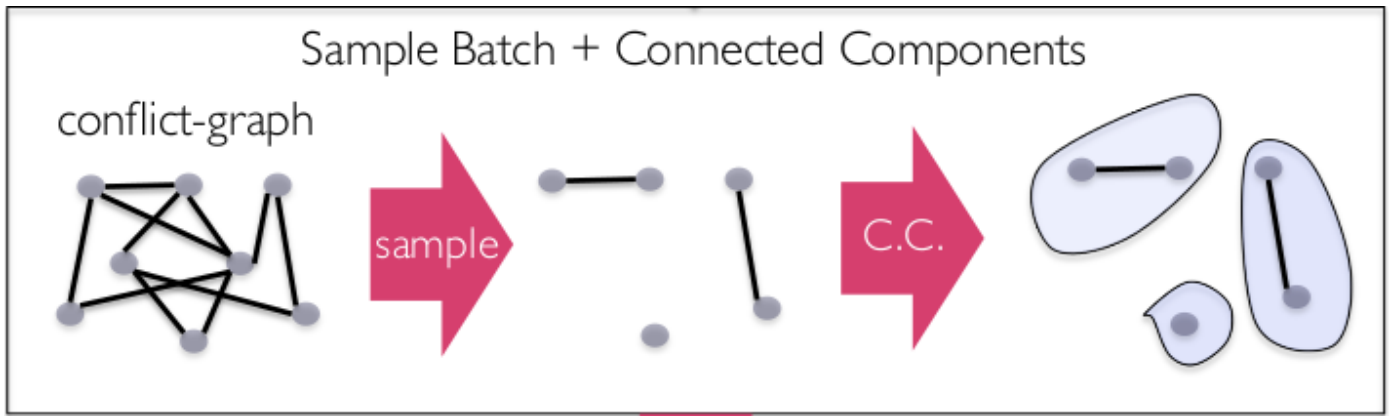
The Updates Conflict Graph

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 1.

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 2.

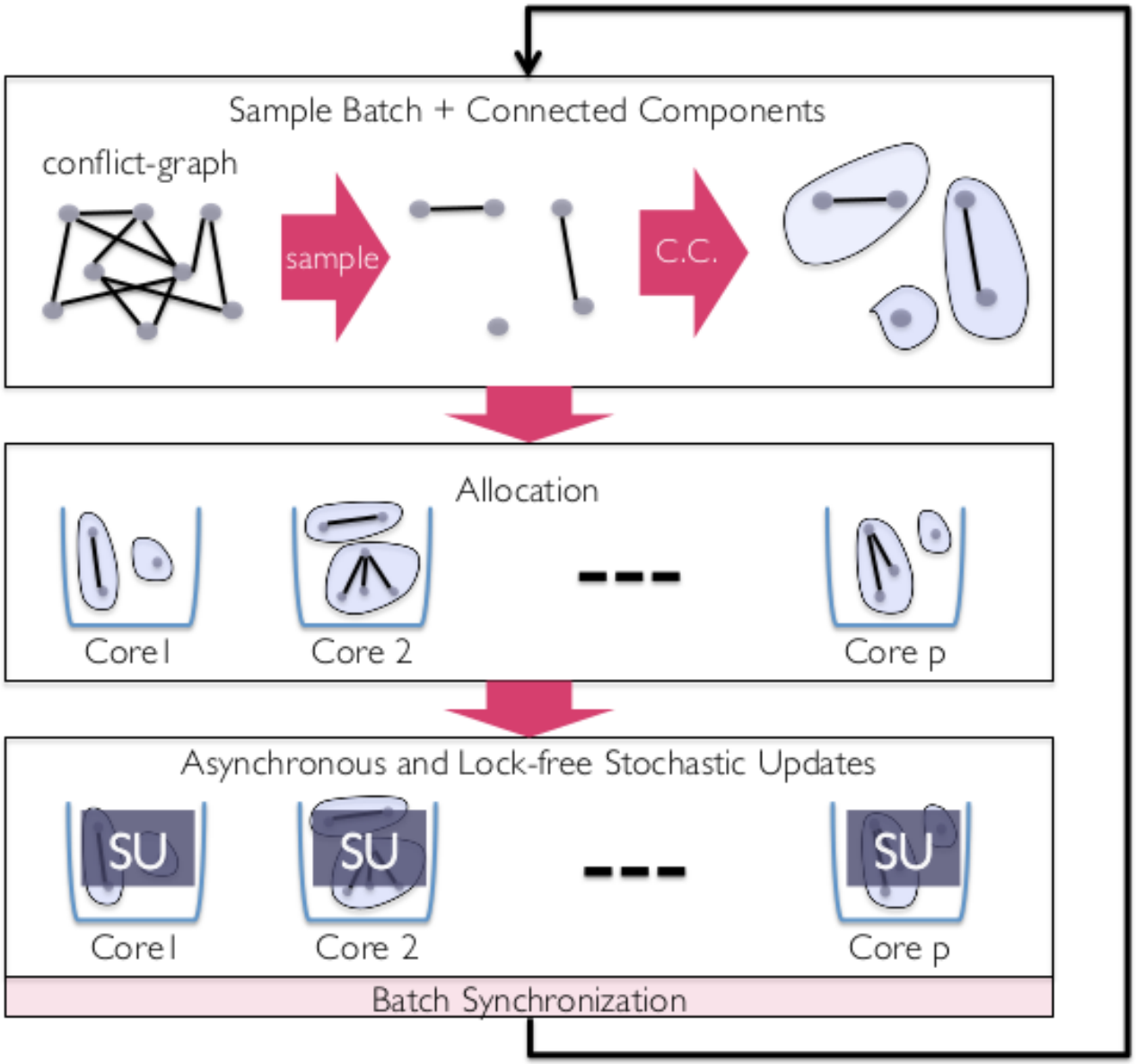
Let G be a graph on n vertices, with maximum vertex degree Δ. Let us sample $B = (1-\varepsilon)\frac{n}{\Delta}$, with or without replacement, and define G' as the induced subgraph on sampled vertices. Then, the largest CC of G' has the size at most $\mathcal{O}\left(\frac{\log n}{\varepsilon^2}\right)$,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

The Number of CCs

Remarks

- Theorem 2 can be regard as a corollary of Theorem 1(Not obvious!). We'll give a brief proof of Theorem 1 later.
- From Theorem 2, 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 4 CYCLADES

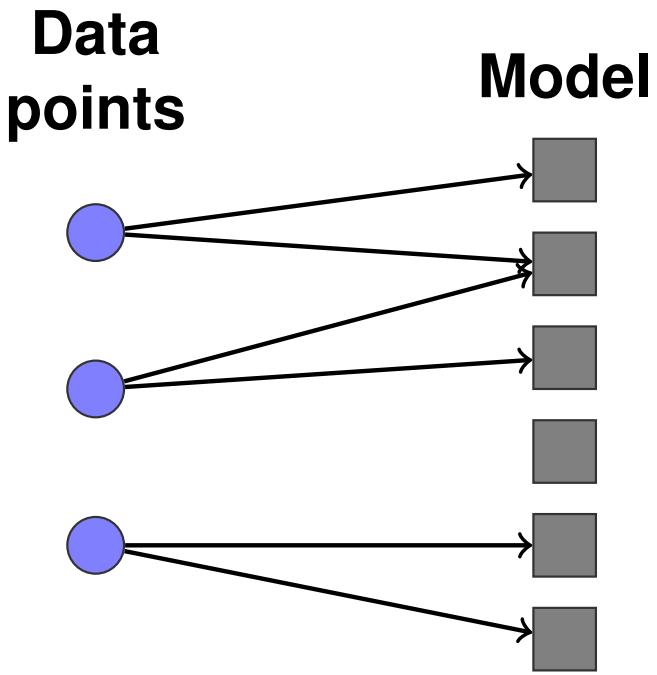
Input: G_u, T, B.
Sample $n_b = T/B$ subgraphs from G_u.
for batch $i = 1 : n_b$ **do**
    Allocate CCs to P cores.
    for each core asynchronously **do**
        Perform local updates in allocated CCs.
    end for
end for

- 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 5 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 3 (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 3

- 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 1 is the most important foundation to develop the main result.