Appendix to "Towards Drug Repositioning: A Unified Computational Framework for Integrating Multiple Aspects of Drug Similarity and Disease Similarity"

(Supplementary for DDR Drug Repositioning Methodology)

Ping Zhang, PhD, Fei Wang, PhD, Jianying Hu, PhD
Healthcare Analytics Research, IBM T.J. Watson Research Center, New York, USA

1. Overview of the BCD Approach for Solving Problem (11)

Since there are multiple groups of variables involved in the optimization problem (11), we adopt an efficient solution based on the *Block Coordinate Descent* (BCD) strategy. The BCD approach works by solving the different groups of variables alternatively until convergence. At each iteration, it solves the optimization problem with respect to one group of variables with all other groups of variables fixed. An overview of the entire BCD procedure is introduced in Algorithm 1.

Algorithm 1: A BCD Approach for Solving Problem (11)

Require: $\lambda_1 \geq 0$, $\lambda_2 \geq 0$, $\delta_1 \geq 0$, $\delta_2 \geq 0$, $K_d > 0$, $K_s > 0$, $\{\boldsymbol{D}_k\}_{k=1}^{K_d}$, $\{\boldsymbol{S}_l\}_{l=1}^{K_s}$, $\boldsymbol{R}$

1: Initialize $\omega=(1/K_d)\mathbf{1} \in \mathbb{R}^{K_d \times 1}$, $\pi=(1/K_s)\mathbf{1} \in \mathbb{R}^{K_s \times 1}$

2: Initialize $\boldsymbol{U}$ and $\boldsymbol{V}$ by performing Symmetric Nonnegative Matrix Factorization on $\tilde{D} = \sum_{k=1}^{K_d} \omega_k D_k$ and $\tilde{S} = \sum_{l=1}^{K_s} \pi_l S_l$ using the method proposed in Wang *et al* (2011).

3: **while** Not Converge **do**

4: Solve $\boldsymbol{\Lambda}$ as described in section 4

5: Solve $\boldsymbol{\Theta}$ as described in section 2

6: Solve ω and π as described in section 3

7: Solve $\mathbf{U}$ as described in section 5

8: Solve $\mathbf{V}$ as described in section 6

9: **end while**

2. Solving Θ

The subproblem of solving Θ is

$$\min_{\Theta} \| \Theta - W \|_F^2 , \text{ subject to } P_{\Omega}(\boldsymbol{\Theta}) = P_{\Omega}(\mathbf{R}) \tag{12}$$

where $\mathbf{W}=\mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^T$. This is a constrained Euclidean projection, and can be decoupled for every element in $\boldsymbol{\Theta}$. Each subproblem has a closed form solution. By aggregating all solutions together, we can get the matrix form representation of the solution as

$$\Theta^* = P_{\Omega}^c(\mathbf{W}) + P_{\Omega}(\mathbf{R}) \tag{13}$$

where Ω^c is the complementary index set for Ω.

3. Solving ω and π

The problems of solving ω and π are similar, so we will just describe how to solve ω as an example. The subproblem of solving ω is

$$\min_{\omega} \sum_{k=1}^{K_d} \omega_k \| D_k - \Sigma \|_F^2 + \delta_1 \| \omega \|_2^2, \text{ subject to } \omega \geqslant 0, \omega^{\mathrm{T}}\mathbf{1}=1 \tag{14}$$

where $\mathbf{\Sigma}=\mathbf{U}\mathbf{U}^{\mathrm{T}}$. Let

$$\boldsymbol{a} = [\| D_1 - \Sigma \|_F^2, \| D_2 - \Sigma \|_F^2, \ldots, \| D_{K_d} - \Sigma \|_F^2]^{\mathrm{T}} \tag{15}$$

Then we can reformulate the problem as

$$\min_{\omega} \delta_1 \left\| \omega - \frac{1}{2\delta_1} \boldsymbol{a} \right\|_2^2 + c, \text{ subject to } \omega \geqslant 0, \omega^{\mathrm{T}}\mathbf{1}=1 \tag{16}$$

where c is some constant irrelevant to ω. This is a standard Euclidean projection problem and can be efficiently solved. In our implementation we used the method in Chen and Ye (2011).

4. Solving Λ

The subproblem of solving $\mathbf{\Lambda}$ is

$$\min_{\Lambda} \| \Theta - U\Lambda V^T \|_F^2, \text{ subject to } \Lambda \geqslant 0 \tag{17}$$

This is a nonnegative quadratic optimization problem and we can use Projected Gradient Descent (PGD) method (Lin 2007) to solve it (for details please refer to section 7). In order to get the gradient of the objective of problem (17) with respect to Λ, we expand it as:

$$J_{\Lambda}=\| \Theta - U\Lambda V^T \|_F^2 = tr(\Theta - \mathrm{U}\Lambda\mathrm{V}^{\mathrm{T}})^{\mathrm{T}}(\Theta - \mathrm{U}\Lambda\mathrm{V}^{\mathrm{T}}) = tr(\mathrm{V}\Lambda^{\mathrm{T}}\mathrm{U}^{\mathrm{T}}\mathrm{U}\Lambda\mathrm{V}^{\mathrm{T}}) - 2tr(\Theta^{\mathrm{T}}\mathrm{U}\Lambda\mathrm{V}^{\mathrm{T}}) + c$$

where c is some constant irrelevant to Λ. Then we can derive the gradient J_{Λ} with respect to Λ as

$$\frac{\partial J_{\Lambda}}{\partial \Lambda} = 2U^T U\Lambda V^T V - 2U^T \Theta V \tag{18}$$

5. Solving U

The subproblem of solving $\mathbf{U}$ is

$$\min_{U} \| \Theta - U\Lambda V^T \|_F^2 + \lambda_1 \sum_{k=1}^{K_d} \omega_k \| D_k - UU^T \|_F^2, \text{ subject to } \mathrm{U} \geqslant 0 \tag{19}$$

We can expand the objective of problem (19) as

$$J_{\mathrm{U}}=\| \Theta - U\Lambda V^T \|_F^2 + \lambda_1 \sum_{k=1}^{K_d} \omega_k \| D_k - UU^T \|_F^2 = tr(U\Lambda V^T V \Lambda^T U^T) - 2tr(U\Lambda V^T \Theta^T) - 2\lambda_1 tr(U^T \tilde{D} U) + \lambda_1 tr(U^T U U^T U) + c$$

where $\tilde{D} = \sum_{k=1}^{K_d} \omega_k D_k$ and c is some constant irrelevant to U. Then the gradient of J_{U} with respect to U is

$$\frac{\partial J_U}{\partial U} = 2U\Lambda V^T V \Lambda^T - 2\Theta V \Lambda^T - 2\lambda_1 \tilde{D} U + 4\lambda_1 U U^T U \tag{20}$$

6. Solving V

The subproblem of solving $\mathbf{V}$ is

$$\min_{V} \| \Theta - U\Lambda V^T \|_F^2 + \lambda_2 \sum_{l=1}^{K_s} \pi_l \| S_l - VV^T \|_F^2, \text{ subject to } \mathrm{V} \geqslant 0 \tag{21}$$

Similarly we can expand the objective of problem (21) as

$$J_V = \| \Theta - U\Lambda V^T \|_F^2 + \lambda_2 \sum_{l=1}^{K_s} \pi_l \| S_l - VV^T \|_F^2 = tr(V\Lambda^T U^T U \Lambda V^T) - 2tr(V^T \Theta^T U \Lambda) - 2\lambda_2 tr(V^T \tilde{S} V) + \lambda_2 tr(V^T V V^T V) + c$$

where $\tilde{S} = \sum_{l=1}^{K_s} \pi_l S_l$ and c is some constant irrelevant to V. Then the gradient of J_V with respect to V is

$$\frac{\partial J_V}{\partial V} = 2V\Lambda^T U^T U \Lambda - 2\Theta^T U \Lambda - 2\lambda_2 \tilde{S} V + 4\lambda_2 V V^T V \tag{22}$$

7. Projected Gradient Descent Method for Optimization

We provide the detail of the Projected Gradient Descent (PGD) method (Lin 2007) here. For notational convenience, we introduce a nonnegative projection operator $P_+(\mathbf{A})$ as

$$(P_+(\boldsymbol{A}))_{ij} = \begin{cases} A_{ij} & \text{if } A_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Then the PG method for solving the problem $\min_{\boldsymbol{A} \geq 0} f(\boldsymbol{A})$ can be presented in Algorithm 2.

Algorithm 2: Projected Gradient

Require: $0<\beta<1$, $0<\sigma<1$. Initialization $\mathbf{A}^{(0)}$.

Ensure: $\mathbf{A}^{(0)} \geqslant 0$

1: **for** $k = 1,2,\ldots$ **do**

2: $\boldsymbol{A}^{(k)} = P_+(\boldsymbol{A}^{(k-1)} - \alpha_k \nabla f(\boldsymbol{A}^{(k-1)}))$, where $\alpha_k = \beta^{t_k}$, and t_k is the first nonnegative integer for which

3: $$f(\boldsymbol{A}^{(k)}) - f(\boldsymbol{A}^{(k-1)}) \leq \sigma \nabla f(\boldsymbol{A}^{(k-1)})^T (\boldsymbol{A}^{(k)} - \boldsymbol{A}^{(k-1)}) \tag{23}$$

4: **end for**

Here condition (23) ensures the sufficient decrease of the function value per iteration, and this rule of determining the stepsize is usually referred to as the Armijo rule (Bertsekas 1976). However, the Armijo rule is usually time consuming, thus we use the following improved PG method in Algorithm 3.

Algorithm 3: Improved Projected Gradient

Require: $0<\beta<1$, $0<\sigma<1$. Initialization $\mathbf{A}^{(0)}$, $\alpha_0=1$.

Ensure: $\mathbf{A}^{(0)} \geqslant 0$

1: **for** $k = 1,2,\ldots$ **do**

2: Assign $\alpha_k = \alpha_{k-1}$

3: **If** α_k satisfies condition (23)

4: repeatedly increase it by $\alpha_k \leftarrow \alpha_k/\beta$ until either α_k does not satisfy (23) or $\mathbf{A}(\alpha_k/\beta)=\mathbf{A}(\alpha_k)$

5: **Else** repeatedly decrease α_k by $\alpha_k \leftarrow \alpha_k . \beta$ until α_k satisfies condition (23)

6: Set $\boldsymbol{A}^{(k)} = P_+(\boldsymbol{A}^{(k-1)} - \alpha_k \nabla f(\boldsymbol{A}^{(k-1)}))$.

7: **end for**

8. Complexity Analysis of the BCD Solution

The computational cost involved in each BCD iteration includes:

- When updating $\boldsymbol{\Theta}$, the main computation happens at calculating $\mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$, which takes $O(nK_dK_s+nmK_s)$ time.
- When updating ω, the main computation happens at calculating $\mathbf{U}\mathbf{U}^{\mathrm{T}}$, which takes $O(n^2K_d)$ time. The Euclidean projection takes $O(K_d \log K_d)$ time.
- When updating π, the main computation happens at calculating $\mathbf{V}\mathbf{V}^{\mathrm{T}}$, which takes $O(m^2K_s)$ time. The Euclidean projection takes $O(K_s \log K_s)$ time.
- Updating $\boldsymbol{\Lambda}$ involves PGD iterations. We just need to evaluate $\mathbf{U}^{\mathrm{T}}\boldsymbol{\Theta}\mathbf{V}$ once, which takes $O(K_dnm+K_dK_sm)$ time. At each iteration evaluating $\mathbf{U}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V}$ takes $O(K_d^2K_s + K_s^2K_d)$ time (as $\mathbf{U}\mathbf{U}^{\mathrm{T}}$ and $\mathbf{V}\mathbf{V}^{\mathrm{T}}$ are already computed), and evaluating the $J_{\boldsymbol{\Lambda}}$ takes $O(K_d^2K_s)$ time.
- Updating $\mathbf{U}$ involves PGD iterations. We just need to evaluate $\boldsymbol{\Theta}\mathbf{V}\boldsymbol{\Lambda}^{\mathrm{T}}$ and $\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V}\boldsymbol{\Lambda}^{\mathrm{T}}$ once, which takes $O(nK_dK_s)$ and $O(K_dK_s^2 + K_sK_d^2)$ time. At each iteration evaluating $\mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V}\boldsymbol{\Lambda}^{\mathrm{T}}$ takes $O(nK_d^2)$ time, $\tilde{D}U$ takes $O(n^2K_d)$, $\mathbf{U}\mathbf{U}^{\mathrm{T}}\mathbf{U}$ takes $O(nK_d^2 + n^2K_d)$ time, and evaluating $J_{\mathbf{U}}$ takes $O(nK_d^2)$ time.
- Updating $\mathbf{V}$ involves PGD iterations. We just need to evaluate $\boldsymbol{\Lambda}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Lambda}$ and $\boldsymbol{\Theta}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Lambda}$ once, which takes $O(nK_dK_s + nK_d^2)$ and $O(mnK_d+mK_dK_s)$ time. At each iteration evaluating $\mathbf{V}\boldsymbol{\Lambda}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Lambda}$ takes $O(mK_s^2)$ time, $\tilde{S}V$ takes $O(m^2K_s)$ time, $\mathbf{V}\mathbf{V}^{\mathrm{T}}\mathbf{V}$ takes $O(mK_s^2 + K_sm^2)$ time, and evaluating $J_{\mathbf{V}}$ takes $O(mK_s^2)$ time.

Adding up everything together, and considering the fact that $\max(K_d,K_s) \ll \min(m,n)$, we can get that the rough computational complexity is $O(R\tilde{r}mn)$, where R is the number of BCD iterations, and $\tilde{r}$ is the average PGD iterations when updating $\boldsymbol{\Lambda}$, $\mathbf{U}$, and $\mathbf{V}$.

Reference

Bertsekas DP (1976) **On the Goldstein-Levitin-Polyak gradient projection method**. *IEEE Transactions on Automatic Control* 21:174–184

Chen Y, Ye X (2011) **Projection onto a simplex**. *arXiv*:1101.6081.

Lin CJ (2007) **Projected gradient methods for nonnegative matrix factorization**. *Neural Comput* 19(10):2756-2779.

Wang F, Li T, Wang X, Zhu S, Ding C (2011) **Community discovery using nonnegative matrix factorization**. *Data Min Knowl Discov* 22: 493–521.