Supplementary material for UniWalk: Explainable and Accurate Recommendation for Rating and Network Data

Haekyu Park * Hyunsik Jeon* Junghwan Kim* Beunguk Ahn* U Kang*

1 Introduction

We propose UniWalk, an explainable and accurate recommender system, in the main paper. This supplementary material contains details skipped in the main paper for better understanding of our method. Section 2 lists symbols used in the paper. Section 3 presents progress of gradient descent for matrix factorization. Section 4 provides details of UniWalk. Section 5 analyzes time complexity. Section 6 presents detailed experimental settings and results.

2 Table of Symbol

Table 1 lists the symbols used in this paper. We denote matrices by upper-case bold letters (e.g. $\mathbf{R}$), vectors by lower-case bold letters (e.g. $\mathbf{z}_v$), scalars by lower-case italic letters (e.g. c), and graphs and sets by upper-case normal letters (e.g. G and U).

3 Gradient Descent for Matrix Factorization

Matrix factorization (MF) learns biases (b_*) and vectors ($\mathbf{x}_*$ and $\mathbf{y}_*$) that approximate predicted ratings to observed ratings. The objective function to minimize is defined in Equation (3.1). r_{ui} is observed ratings of a user u and an item i, μ is global average rating, b_u is u's bias, b_i is i's bias, $\mathbf{x}_u$ is u's vector, $\mathbf{y}_i$ is i's vector, λ is regularization parameter, and K is the set of (u, i) pairs where r_{ui} is observed.

$$L = \frac{1}{2} \sum_{(u,i) \in K} (r_{ui} - \mu - b_u - b_i - \mathbf{x}_u^T \mathbf{y}_i)^2 + \lambda (b_u^2 + b_i^2 + ||\mathbf{x}_u||^2 + ||\mathbf{y}_i||^2) \tag{3.1}$$

Gradient descent is used to minimize L. The update procedures for parameters are as follows, where η is learning rate.

$$b_u \leftarrow b_u - \eta \frac{\partial L}{\partial b_u}, \quad \frac{\partial L}{\partial b_u} = -e_{ui} + \lambda b_u$$

$$b_i \leftarrow b_i - \eta \frac{\partial L}{\partial b_i}, \quad \frac{\partial L}{\partial b_i} = -e_{ui} + \lambda b_i$$

Algorithm 1 Overall Process of UniWalk

Input: Rating $\mathbf{R}$ and social network $G = (V, E)$
Output: Recommendation results and explanation

1: Build a unified graph $\tilde{G} = (\tilde{V}, \tilde{E})$.
2: **Learn embedding of entities by Algorithm 2.**
3: Recommend items with explanation for each user.
4: **return** predicted rating matrix $\hat{\mathbf{R}}$

$$\mathbf{x}_u \leftarrow \mathbf{x}_u - \eta \nabla_{\mathbf{x}_u} L, \quad \nabla_{\mathbf{x}_u} L = -e_{ui} \mathbf{y}_i + \lambda \mathbf{x}_u$$

$$\mathbf{y}_i \leftarrow \mathbf{y}_i - \eta \nabla_{\mathbf{y}_i} L, \quad \nabla_{\mathbf{y}_i} L = -e_{ui} \mathbf{x}_u + \lambda \mathbf{y}_i$$

, where η is learning rate and e_{ui} is a prediction error defined as $e_{ui} = r_{ui} - \mu - b_u - b_i - \mathbf{x}_u^T \mathbf{y}_i$.

4 Details of UniWalk

The overall process of UniWalk is summarized in Algorithm 1. This section focuses on the second step with detailed explanations. Step 2 for network embedding consists of two phases: sampling phase and optimization phase. Sampling phase generates node sequences with random walks to sample desired entity pairs for $\mathcal{D}^R$, $\mathcal{D}^+$, and $\mathcal{D}^-$. Optimization phase learns factors of entities with the sampled node sequences. Section 4.1 presents progress of learning parameters in optimization phase. Section 4.2 states detailed algorithms in the two phases.

4.1 Optimization Phase of UniWalk

UniWalk learns biases (b_*) and vectors ($\mathbf{z}_*$) of entities that minimize the objective function defined in Equation (4.2) and (4.3). $\hat{r}_{ui}$ is a predicted rating for a user u and an item i and is calculated as $\mu + b_u + b_i + \mathbf{z}_u^T \mathbf{z}_i$. λ_b and λ_z are regularization parameters for biases and vectors, respectively. α and β are weight of the positive unsupervised term and the negative unsupervised term, respectively. $\mathcal{D}^R$ is a set of entity pairs connected by ratings, $\mathcal{D}^+$ is that of similar entities, and $\mathcal{D}^-$ is that of dissimilar entities.

*Seoul National University, Seoul, Republic of Korea. {hkpark627, jeon185, kjh900809, elaborate, ukang}@snu.ac.kr

Copyright © 2018 by SIAM
Unauthorized reproduction of this article is prohibited

Table 1: Table of symbols.

Symbol	Definition
U	set of users who give rating
I	set of rated items
u	user
i	item
$\mathbf{R} \in \mathbb{R}^{\lvert U\rvert \times \lvert I\rvert}$	observed rating matrix
r_{ui}	observed rating of u on i
$\hat{\mathbf{R}} \in \mathbb{R}^{\lvert U\rvert \times \lvert I\rvert}$	predicted rating matrix
d	dimension of embedding vectors
$\hat{r}_{ui}$	predicted rating of u on i
μ	average of ratings
$G = (V, E)$	social graph of users
$\tilde{G} = (\tilde{V}, \tilde{E})$	unified graph. $\tilde{V} = U \cup I \cup V$
v, $w \in \tilde{V}$	nodes or entities
$\mathbf{b} \in \mathbb{R}^{\lvert\tilde{V}\rvert}$	bias vector of all entities
b_v	bias of v
$\mathbf{Z} \in \mathbb{R}^{\lvert\tilde{V}\rvert \times d}$	latent vector matrix of entities
$\mathbf{z}_v \in \mathbb{R}^d$	embedding vector of v
c	weight of social edges in $\tilde{G}$
l	length of a random walk
s	size of sliding window
$\mathcal{D}^R \subseteq U \times I$	multiset of (u, i) linked by ratings
$\mathcal{D}^+ \subseteq \tilde{V} \times \tilde{V}$	multiset of pairs of similar entities
$\mathcal{D}^- \subseteq \tilde{V} \times \tilde{V}$	multiset of pairs of dissimilar entities
α	weight of positive unsupervised term
β	weight of negative unsupervised term
λ_z	regularization parameter of vectors
λ_b	regularization parameter of biases
η	learning rate
γ	momentum parameter

$$L = \overbrace{\sum_{(u,i)\in\mathcal{D}^R} \frac{1}{2}(r_{ui} - \hat{r}_{ui})^2}^{\text{Supervised term}} + \overbrace{\frac{\lambda_b}{2}||\mathbf{b}||^2 + \frac{\lambda_z}{2}||\mathbf{Z}||_F^2}^{\text{Regularization term}} \tag{4.2}$$

$$+\alpha \overbrace{\cdot \sum_{(v,w)\in\mathcal{D}^+} -\mathbf{z}_v^T \mathbf{z}_w}^{\text{Positive unsupervised term}} + \beta \overbrace{\cdot \sum_{(v,w)\in\mathcal{D}^-} \mathbf{z}_v^T \mathbf{z}_w}^{\text{Negative unsupervised term}} \tag{4.3}$$

In optimization phase of UniWalk, we use gradient descent with momentum to update parameters. The momentum term leads to a faster convergence of the parameters. The momentum update for a parameter θ is given in Equation (4.4), where v_t is velocity vector at time t, and γ is a momentum parameter for how much the previous gradients are incorporated into the current update.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{4.4}$$

For a node pair $(v, w) \in \mathcal{D}^R$, we update biases b_v and b_w, and latent vectors $\mathbf{z}_v$ and $\mathbf{z}_w$ to minimize the supervised term (Equation (4.2)). We use Equation (4.5) to update b_v, where $e_{vw} = r_{v,w} - (\mu + b_v + b_w + \mathbf{z}_v^T \mathbf{z}_w)$ and $r_{v,w}$ is rating betwen v and w.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta(-e_{vw} + \lambda_b b_w) \\ b_v &= b_v - v_t \end{aligned} \tag{4.5}$$

To update b_w, we use Equation (4.6).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta(-e_{vw} + \lambda_b b_v) \\ b_w &= b_w - v_t \end{aligned} \tag{4.6}$$

To update $\mathbf{z}_v$ we use Equation (4.7).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta(-e_{vw}\mathbf{z}_w + \lambda_z \mathbf{z}_v) \\ \mathbf{z}_v &= \mathbf{z}_v - v_t \end{aligned} \tag{4.7}$$

To update $\mathbf{z}_w$ we use Equation (4.8).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta(-e_{vw}\mathbf{z}_v + \lambda_z \mathbf{z}_w) \\ \mathbf{z}_w &= \mathbf{z}_w - v_t \end{aligned} \tag{4.8}$$

For a node pair $(v, w) \in \mathcal{D}^+$, we update $\mathbf{z}_v$ and $\mathbf{z}_w$, where α is the weight of the positive unsupervised term (left term in Equation (4.3)). To update $\mathbf{z}_v$, we use Equation (4.9).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta\alpha(-\mathbf{z}_w + \lambda_z \mathbf{z}_v) \\ \mathbf{z}_v &= \mathbf{z}_v - v_t \end{aligned} \tag{4.9}$$

To update $\mathbf{z}_w$ we use Equation (4.10).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta\alpha(-\mathbf{z}_v + \lambda_z \mathbf{z}_w) \\ \mathbf{z}_w &= \mathbf{z}_w - v_t \end{aligned} \tag{4.10}$$

For a node pair $(v, w) \in \mathcal{D}^-$, we update $\mathbf{z}_v$ and $\mathbf{z}_w$, where β is the weight of the negative unsupervised term (right term in Equation (4.3)). To update $\mathbf{z}_v$, we use Equation (4.11).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta\beta(+\mathbf{z}_w + \lambda_z \mathbf{z}_v) \\ \mathbf{z}_v &= \mathbf{z}_v - v_t \end{aligned} \tag{4.11}$$

To update $\mathbf{z}_w$ we use Equation (4.12).

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta\beta(+\mathbf{z}_v + \lambda_z \mathbf{z}_w) \\ \mathbf{z}_w &= \mathbf{z}_w - v_t \end{aligned} \tag{4.12}$$

All the velocity vectors in Equation from (4.5) to (4.12) are distinct.

4.2 Algorithms in Step 2 of UniWalk

We state algorithm of network embedding of UniWalk, which is executed in the second line in Algorithm 1. Given a unified graph $\tilde{G} = (\tilde{V}, \tilde{E})$, network embedding learns biases and vectors of all entities in $\tilde{V}$. Network embedding consists of two phases: sampling and optimization. In the sampling phase, UniWalk samples node sequences with positive, negative, and unweighted random walks defined in DEFINITION 3 of the main paper. In the optimization phase, UniWalk learns the factors of entities with the sampled walks that minimize the objective function.

Copyright © 2018 by SIAM
Unauthorized reproduction of this article is prohibited

Algorithm 2 Network embedding

Input: Unified graph $\tilde{G} = (\tilde{V}, \tilde{E})$, length of walk l, and size of window s

Output: Embedding vector $\mathbf{z}_v$, biases b_v for all $v \in \tilde{V}$, and pair sets $\mathcal{D}^R$ and $\mathcal{D}^+$ for explanation

1: Initialize $\mathcal{D}^R = \{\}$, $\mathcal{D}^+ = \{\}$
2: **while** RMSE not converged **do**
3: // Generate node sequences
4: $walks^0 = GenWalks(\tilde{G}, l, 0)$
5: $walks^+ = GenWalks(\tilde{G}, l, +)$
6: $walks^- = GenWalks(\tilde{G}, l, -)$
7: // Learn factors with the walks
8: **while** RMSE not converged **do**
9: $\mathcal{D}^R, \mathcal{D}^+ = LearnWalks(walks^0, s, \mathcal{D}^R, \mathcal{D}^+, 0)$
10: $\mathcal{D}^R, \mathcal{D}^+ = LearnWalks(walks^+, s, \mathcal{D}^R, \mathcal{D}^+, +)$
11: $\mathcal{D}^R, \mathcal{D}^+ = LearnWalks(walks^-, s, \mathcal{D}^R, \mathcal{D}^+, -)$
12: **end while**
13: **end while**
14: **return** $\mathbf{z}_v$ and b_v for all $v \in \tilde{V}$, $\mathcal{D}^R$, and $\mathcal{D}^+$

The overall process of the network embedding is shown in Algorithm 2. In the sampling phase (lines 4-6 in Algorithm 2), node sequences are generated by *GenWalks* (Algorithm 3). In the optimization phase (lines 10-11 in Algorithm 2), node pairs are extracted, and the pairs are used to update parameters using gradient descent with momentum in *LearnWalks* (Algorithm 4). $\mathcal{D}^R$ and $\mathcal{D}^+$ are returned for recommendation explanations. `for` statement in line 4 of *LearnWalks* generates *neighbors* to be paired up with a *target* node. The parameter updates are executed according to types of (*target*, *neighbor*) pairs. Algorithm 2 returns only $\mathcal{D}^R$ and $\mathcal{D}^+$, because they are used in explanation while $\mathcal{D}^-$ are not. Node pairs for $\mathcal{D}^-$ are not saved, but they are extracted from the negative walks and used in learning (lines 17-18 in Algorithm 4).

5 Time Complexity of UniWalk for Rating Prediction

UniWalk takes $O(|\tilde{E}| + lsd|\tilde{V}|)$ time for rating prediction, where $\tilde{G} = (\tilde{V}, \tilde{E})$ is the unified graph, l is the length of a random walk, and s is the size of window. We prove the time complexity as follows.

THEOREM 5.1. *The time complexity of UniWalk is* $O(|\tilde{E}| + lsd|\tilde{V}|)$.

Proof. UniWalk for rating prediction consists of three steps: building a unified graph (step 1), graph embedding on the graph (step 2), and predicting unobserved ratings (step 3). Step 1 takes $O(|\tilde{E}|)$ time according to Lemma 5.1. Step 2 takes $O(lsd|\tilde{V}|)$ time according to Lemma 5.2. Step 3 requires $O(|\tilde{E}|)$ times because we predict ratings in a test set whose size is less than $O(|\tilde{E}|)$. Therefore, the total steps of UniWalk take $O(|\tilde{E}| + lsd|\tilde{V}|)$.

Algorithm 3 Generate walks (*GenWalks*)

Input: A graph $\tilde{G} = (\tilde{V}, \tilde{E})$, length of walk l, and type of walk

Output: Node sequences *walks*

1: *walks* = [ ]
2: **for** $v \in \tilde{V}$ **do**
3: **if** Type of walk is + **then**
4: Sample a node sequence w_v starting from v
5: by positive random walk of length l.
6: **else if** Type of walk is - **then**
7: Sample a node sequence w_v starting from v
8: by negative random walk of length l.
9: **else if** Type of walk is 0 **then**
10: Sample a node sequence w_v starting from v
11: by unweighted random walk of length l.
12: **end if**
13: Append w_v to *walks*
14: **end for**
15: **return** *walks*

LEMMA 5.1. *Building a unified graph* $\tilde{G} = (\tilde{V}, \tilde{E})$ *corresponding to step 1 takes* $O(|\tilde{E}|)$ *time.*

Proof. Making user-item edges with a rating matrix $\mathbf{R}$ takes $O(nnz(\mathbf{R}))$, where $nnz(\mathbf{R})$ is the number of observed ratings. Making user-user edges with a social network $G = (V, E)$ takes $O(|E|)$. $|\tilde{E}| = |E| + nnz(\mathbf{R})$ because an edge in $\tilde{E}$ is either rating or social link. Therefore, the time complexity of step 1 is $O(nnz(\mathbf{R}) + |E|) = O(|\tilde{E}|)$.

LEMMA 5.2. *Network embedding on the unified graph corresponding to step 2 takes* $O(lsd|\tilde{V}|)$ *time.*

Proof. Step 2 consists of two phases: sampling and optimization phase. The sampling phase requires $O(l|\tilde{V}|)$ time, because random walks generate $|\tilde{V}|$ node sequences of length l. The optimization phase takes $O(lsd|\tilde{V}|)$ time, because *LearnWalks* (Algorithm 4) takes $O(lsd|\tilde{V}|)$ time for all types of walks. We have $O(|\tilde{V}|)$ node sequences. Each sequence generates $O(ls)$ pairs, because we have $O(l)$ target node and $O(s)$ neighbors for each target node. Each parameter update and saving pairs take $O(d)$. Therefore, *LearnWalks* takes $O(|\tilde{V}|) \times O(ls) \times O(d) = O(lsd|\tilde{V}|)$ time. Finally, the total complexity of step 2 is $O(l|\tilde{V}|) + O(lsd|\tilde{V}|) = O(lsd|\tilde{V}|)$.

6 Experiment

We provide details of experiments. We present the definition of RMSE (Root Mean Squared Error) and MAE (Mean Average Error) to measure accuracy. We

Copyright © 2018 by SIAM
Unauthorized reproduction of this article is prohibited

Algorithm 4 Learn from random walks ($LearnWalks$)

Input: Node sequences $walks$, size of a window s, set of rating pairs $\mathcal{D}^R$, set of similar nodes $\mathcal{D}^+$, and type of walk

Output: Sets of accumulated node pairs $\mathcal{D}^R$ and $\mathcal{D}^+$

```
for a walk ∈ walks do
    while a window W sliding in walk do
        t = target node in the center of W
        for v ∈ W which is not equal to t do
            if (t, v) is linked by a rating then
                // (t, v) is in D^R
                Update parameters for (t, v) with
                  Equation from (4.5) to (4.8)
                D^R = D^R ∪ {(t, v)}
            else if Type of walk is + then
                // (t, v) is in D^+
                Update parameters for (t, v) with
                  Equation (4.9) and (4.10)
                D^+ = D^+ ∪ {(t, v)}
            else if Type of walk is - then
                // (t, v) is in D^-
                Update parameters for (t, v) with
                  Equation (4.11) and (4.12)
            end if
        end for
    end while
end for
return D^R and D^+
```

report hyperparameter settings of UniWalk, UCF, ICF, and MF. We show additional experiments to show hyperparameter sensitivity and scalability of UniWalk.

6.1 Metrics for Accuracy. For each user u and item i, RMSE and MAE are defined as follows, where K is a set of user-item pairs for which r_{ui} is observed.

- RMSE (Root Mean Square Error)

$$RMSE = \sqrt{\frac{1}{|K|}\sum_{(u,i)\in K}(r_{ui}-\hat{r}_{ui})^2} \tag{6.13}$$

- MAE (Mean Absolute Error) .

$$MAE = \frac{1}{|K|}\sum_{(u,i)\in K}|r_{ui}-\hat{r}_{ui}|. \tag{6.14}$$

6.2 Hyperparameter Settings. We use hyperparameters for UniWalk as follows. In *Filmtrust* dataset, c=5, l=30, α=0.05, β=0.005, d=25, s=7, λ_b=0.1, λ_z=0.1, η=0.01, and γ=0.2. In *Epinions* dataset, c=6, l=50, α=0.001, β=0.0007, d=25, s=7, λ_b=0.08, λ_z=1.3, η=0.003, and γ=0.2. In *Flixster* dataset, c=5, l=50, α=0.001, β=0.001, d=25, s=7, λ_b=0.2, λ_z=0.2, η=0.005, and γ=0.2.

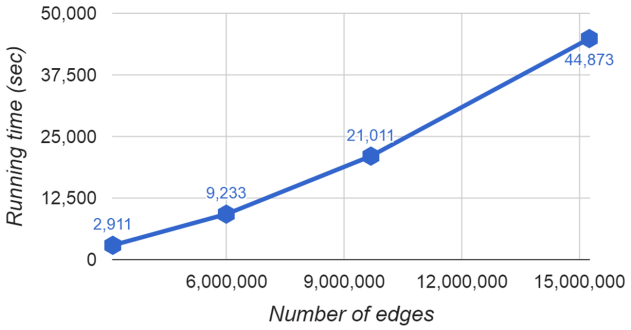


Figure 1: Running time of UniWalk with varying number of edges in a unified graph of *Flixster* dataset. The running time increases near linearly with regard to the number of edges.

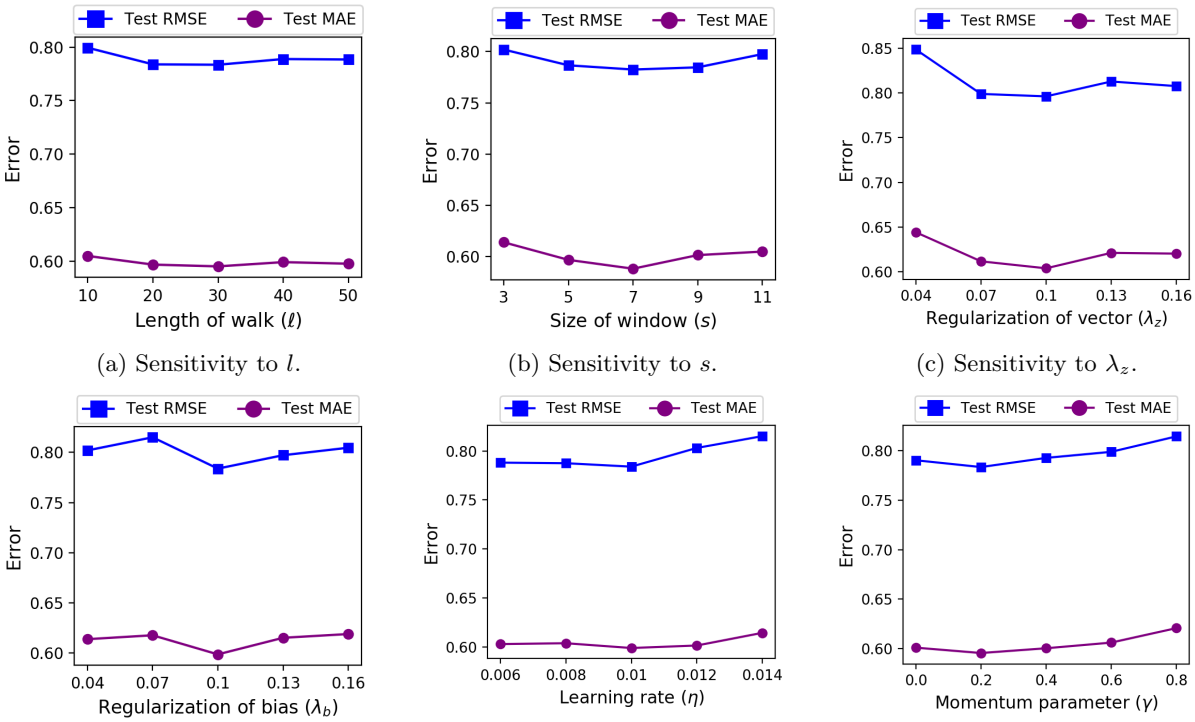
We use hyperparameters for UCF and ICF as follows. In *Filmtrust* dataset, k=3. In *Epinions* dataset, k=5. In *Flixster* dataset, k=3.

We use hyperparameters for MF as follows. In *Filmtrust* dataset, λ=0.75, and d=5. In *Epinions* dataset, λ=1, and d=5. In *Flixster* dataset, λ=5, and d=5.

6.3 Hyperparameter sensitivity. We study the sensitivity of UniWalk with respect to the following hyperparameters: length of random walks (l), size of sliding window (s), regularization parameter of vectors (λ_z), regularization parameter of biases (λ_b), learning rate (η), and momentum parameter (γ). We perform the experiment on *FilmTrust* dataset. We measure RMSE and MAE of UniWalk while varying the value of one hyperparameter at a time. The results are presented in Figure 2. l and s do not affect the performance of the model significantly as seen in Figure 2a and 2b. Other hyperparameters affect the accuracy of UniWalk as seen in Figure 2c, 2d, 2e, and 2f. For the momentum parameter γ, the performance degrades as the γ increases.

6.4 Scalability of UniWalk. We show the scalability of UniWalk by measuring running times with varying number of edges in a unified graph of *Flixster* dataset, the largest one in our experiments. Running time of UniWalk increases near linearly with the number of edges as seen in Figure 1. The linear tendency in running time indicates that UniWalk is scalable with the input data size, which is in accordance with the time complexity of UniWalk stated in Section 5. The running time with varying number of nodes also exhibits a linear tendency similar to that with varying number of edges.

Copyright © 2018 by SIAM
Unauthorized reproduction of this article is prohibited



(a) Sensitivity to l. (b) Sensitivity to s. (c) Sensitivity to λ_z.

(d) Sensitivity to λ_b. (e) Sensitivity to η. (f) Sensitivity to γ.

Figure 2: UniWalk's hyperparameter sensitivity.

Copyright © 2018 by SIAM
Unauthorized reproduction of this article is prohibited