Algorithm 1: identify Row Context

Input: r_i, $Backgrd(T_i)$=$T_1, T_2, \ldots, T_n$ and similarity threshold θ_r

Output: $con(r_i)$

$con(r_i) = \Phi$;
for $j = 1; j \leq n; j \neq i$ **do**
    float $maxSim = 0$;
    $r^{maxSim} = null$;
    while *not end of* T_j **do**
        compute Jaro$(r_i, r_m)(r_m \in T_j)$;
        if $(Jaro(r_i, r_m) \geq \theta_r) \wedge (Jaro(r_i, r_m) \geq r^{maxSim})$ **then**
            replace r^{maxSim} with r_m;
    $con(r_i) = con(r_i) \cup r^{maxSim}$;
return $con(r_i)$;

Algorithm 2: Service checkpoint image storage node and routing path selection

Input: host server PM_s that $SerImg_k$ is fetched from, $subnet_s$ that PM_s belongs to, pod_s that PM_s belongs to

Output: Service image storage server *storageserver*,and the image transfer path *path*

storageserver = Storage node selection(PM_s, $SerImg_k$,$subnet_s$,pod_s);
if *storageserver* $\neq$ *null* **then**
    select a path from *storageserver* to PM_s and assign the path to *path*;
final ;
return *storageserver* and *path*;

Algorithm 3: Storage node selection

Input: host server PM_s that the checkpoint image Img is fetched from, $subnet_s$ that PM_s belongs to, pod_s that PM_s belongs to

Output: Image storage server $storageserver$

for *each host server* PM_i *in the same subnet with* PM_s **do**
    if PM_i *is not a service providing node or checkpoint image storage node of* S_k **then**
        add PM_i to $candidateList$;
sort $candidateList$ by reliability desc;
init $storageserver$; **for** *each* PM_k *in* $candidateList$ **do**
    if $SP(PM_k) \geq E(SP)$ *of* pod_i *and* $BM_k \leq$ *size of* Img **then**
        assign PM_k to $storageserver$;
        goto final;
clear $candidateList$;
add all other subnets in pod_s to $netList$;
for *each subnet* $subnet_j$ *in* $netList$ **do**
    clear $candidateList$;
    for *each* PM_i *in* $subnet_j$ **do**
        if PM_i *is not a service providing node or checkpoint image storage node of* S_k **then**
            add PM_i to $candidateList$;
    sort all host in $candidateList$ by reliability desc;
    for *each* PM_k *in* $candidateList$ **do**
        if $SP(PM_k) \geq E(SP)$ *of* pod_i *and* $BM_k \leq$ *size of* Img **then**
            assign PM_k to $storageserver$;
            goto final;
final ;
return $storageserver$;

Algorithm 4: component matrices computing

Input: $\mathcal{X} \in \mathbb{R}^{l_1 \times l_2 \times \cdots \times l_N}, \varepsilon, \lambda, \delta, R$
Output: $A^{(j)}s$ for $j = 1$ to N

Initialize all $A^{(j)}s$ //which can be seen as the 0^{th} round iterations;
$l'' = L$ //if we need to judge whether (11) is true then l'' denotes $L|_{t-1}$;
for *each* $A^j_{i_j r} (1 \leq j \leq N, 1 \leq i_j \leq I_j, 1 \leq r \leq R)$ **do**
    //1^{st} round iterations;
    $g^{(j)'}_{i_j r} = g^{(j)}_{i_j r}$;
    $A^{(j)'}_{i_j r} = A^{(j)}_{i_j r}$//if the rollback shown as (12) is needed,$A^{(j)'}_{i_j r}$ denotes $A^{(j)}_{i_j r}|_{t-1}$;
    $A^{(j)}_{i_j r} = A^{(j)}_{i_j r} - \mathbf{sign}\left(g^{(j)}_{i_j r}\right) \cdot \delta^{(j)}_{i_j r}$;
repeat//other rounds of iterations for computing component matrices
    $l' = L$ //if we need to judge whether (11) is true then l' denotes $L|_t$;
    for *each* $A^j_{i_j r} (1 \leq j \leq N, 1 \leq i_j \leq I_j, 1 \leq r \leq R)$ **do**
        if $g^{(j)}_{i_j r} \cdot g^{(j)'}_{i_j r} > 0$ **then**
            $A^{(j)'}_{i_j r} = A^{(j)}_{i_j r}$;
            $g^{(j)'}_{i_j r} = g^{(j)}_{i_j r}$;
            $\delta^{(j)}_{i_j r} = \mathbf{min}\left(\delta^{(j)}_{i_j r} \cdot \eta^{+}, Max\_Step\_Size\right)$;
            $A^{(j)}_{i_j r} = A^{(j)}_{i_j r} - \mathbf{sign}\left(g^{(j)}_{i_j r}\right) \cdot \delta^{(j)}_{i_j r}$;
        else if $g^{(j)}_{i_j r} \cdot g^{(j)'}_{i_j r} < 0$ **then**
            if $l' > l''$ **then**
                $g^{(j)'}_{i_j r} = g^{(j)}_{i_j r}$;
                $A^{(j)}_{i_j r} = A^{(j)'}_{i_j r}$// if (11) is true then rollback as (12);
                $\delta^{(j)}_{i_j r} = \mathbf{max}\left(\delta^{(j)}_{i_j r} \times \eta^{-}, Min\_Step\_Size\right)$;
            else
                $A^{(j)'}_{i_j r} = A^{(j)}_{i_j r}$;
                $g^{(j)'}_{i_j r} = g^{(j)}_{i_j r}$;
                $\delta^{(j)}_{i_j r} = \mathbf{max}\left(\delta^{(j)}_{i_j r} \cdot \eta^{-}, Min\_Step\_Size\right)$;
                $A^{(j)}_{i_j r} = A^{(j)}_{i_j r} - \mathbf{sign}\left(g^{(j)}_{i_j r}\right) \cdot \delta^{(j)}_{i_j r}$;
        else
            $A^{(j)'}_{i_j r} = A^{(j)}_{i_j r}$;
            $g^{(j)'}_{i_j r} = g^{(j)}_{i_j r}$;
            $A^{(j)}_{i_j r} = A^{(j)}_{i_j r} - \mathbf{sign}\left(g^{(j)}_{i_j r}\right) \cdot \delta^{(j)}_{i_j r}$;
    $l'' = l'$;
until $\boldsymbol{L \leq \varepsilon}$ *or maximum iterations exhausted*;

Algorithm 5: Learning algorithm of R2P

Input: ratings R, joint demographic representations Y,learning rate η,maximum iterative number $maxIter$, negative sampling number k;

Output: interaction matrix $\boldsymbol{W}$, movie vectors V;

Initialize $\boldsymbol{W}, V$ randomly;
$t = 0$;
For convenience, define $\vec{\varphi}_n = \sum_{m \in S_n} r_{m,n} \vec{v}_m$;
while *not converged* or $t > maxIter$ **do**
    t = t+1;
    for $n = 1; n \leq N; n++$ **do**
        $\boldsymbol{W} = \boldsymbol{W} + \eta(1 - \sigma\left(\vec{\varphi}_n^T \boldsymbol{W} \vec{y}_n\right))\vec{\varphi}_n \vec{y}_n^T$;
        for $m \in S_n$ **do**
            $\vec{v}_m = \vec{v}_m + \eta\left(1 - \sigma\left(\vec{\varphi}_n^T \boldsymbol{W} \vec{y}_n\right)\right) r_{m,n} \boldsymbol{W} \vec{y}_n$;
        for $i = 1; i \leq k; i++$ **do**
            sample negative sample $\vec{y}_i$ from P_n;
            $\boldsymbol{W} = \boldsymbol{W} - \eta(1 - \sigma\left(-\vec{\varphi}_n^T \boldsymbol{W} \vec{y}_n\right))\vec{\varphi}_n \vec{y}_i^T$;
            for $m \in S_n$ **do**
                $\vec{v}_m = \vec{v}_m - \eta\left(1 - \sigma\left(-\vec{\varphi}_n^T \boldsymbol{W} \vec{y}_n\right)\right) r_{m,n} \boldsymbol{W} \vec{y}_i$;
    $\boldsymbol{W} = \boldsymbol{W} - 2\lambda\eta\boldsymbol{W}$;
    $V = V - 2\lambda\eta V$
return $\boldsymbol{W}, V$;