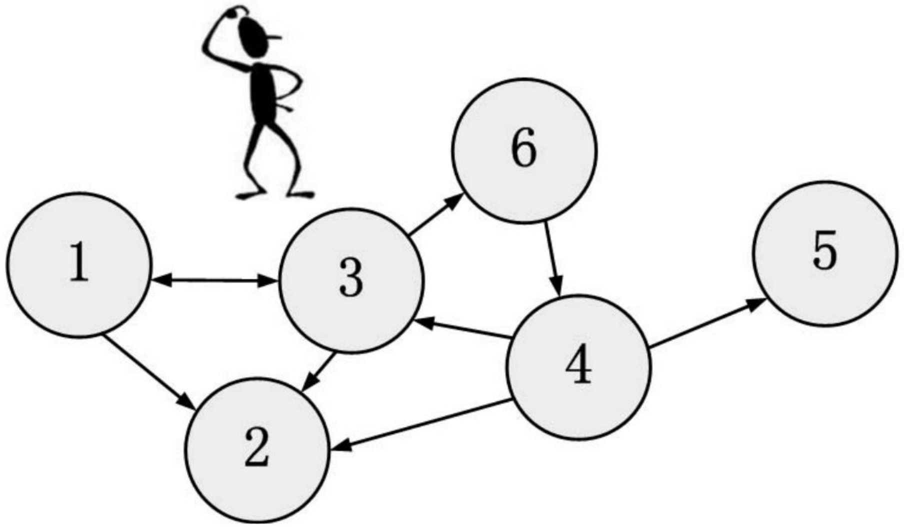
Accelerating Topic-Sensitive PageRank By Exploiting the Query History

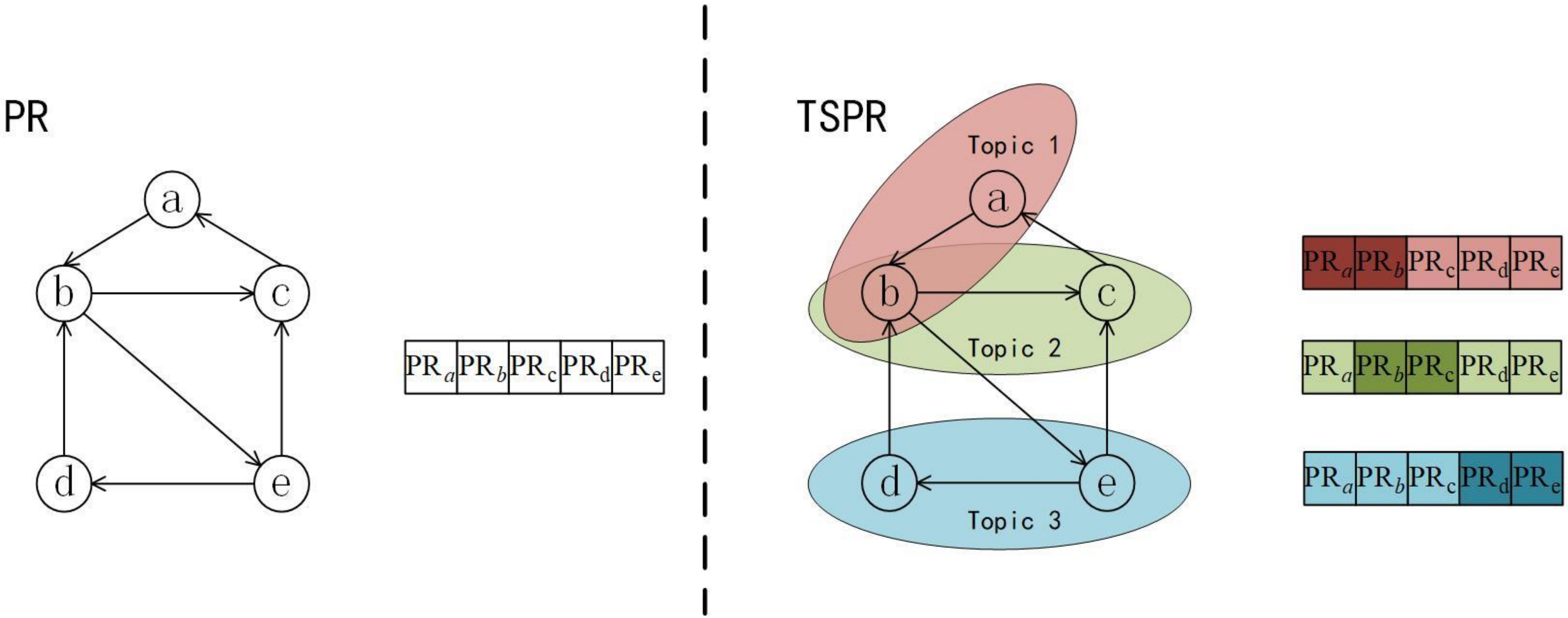
Shufeng Gong, **Zhixin Zhang**, YanFeng Zhang, Cong Fu, and Ge Yu
Northeastern University

PageRank



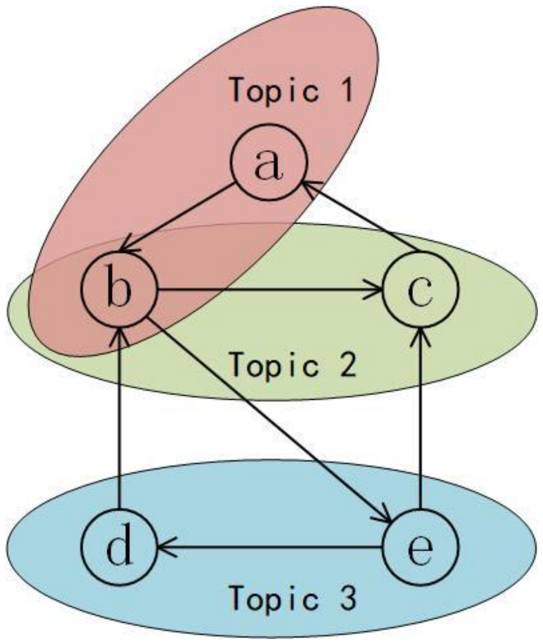
Node	PageRank
1	0.1296
2	0.2378
3	0.1853
4	0.1873
5	0.1302
6	0.1296

Topic Sensitive PageRank



Power Iteration

- $\vec{S}$: PageRank score vector
- P : transition matrix of graph
- $\vec{e}$: indicator topic vector
- α : teleportation constant



first iteration:

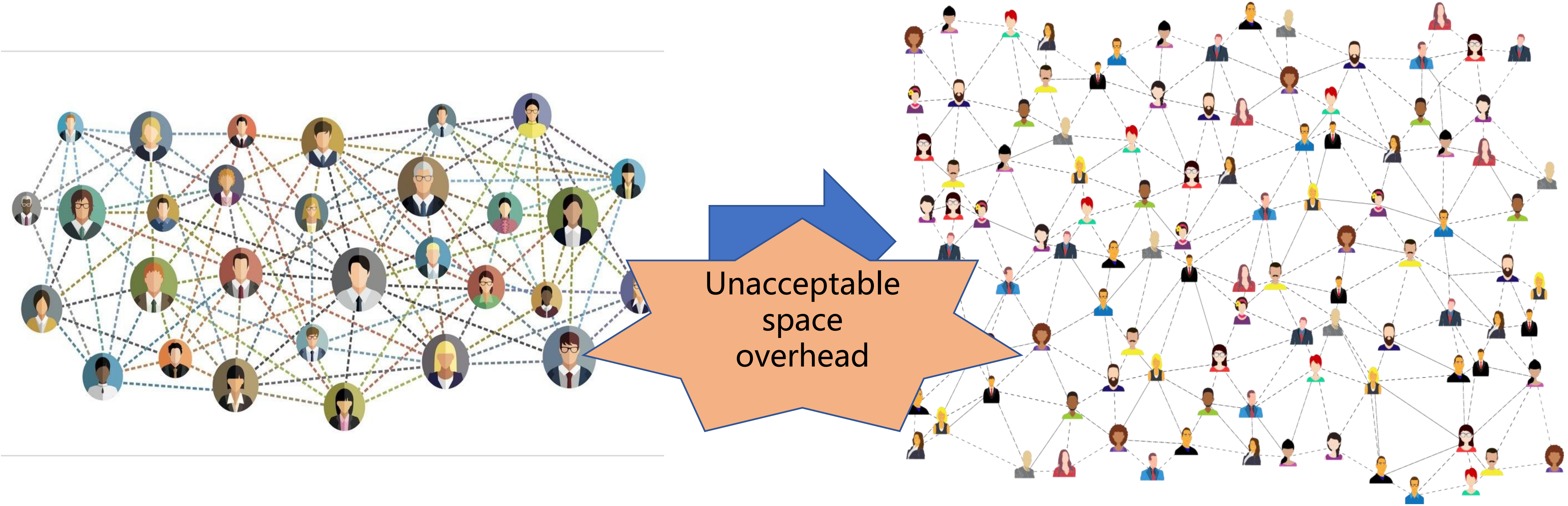
$$\underbrace{\begin{bmatrix} s_0^{(1)} \\ s_1^{(1)} \\ s_2^{(1)} \\ s_3^{(1)} \\ s_4^{(1)} \end{bmatrix}}_{\vec{S}^{(1)}} = (1-\alpha)\cdot \underbrace{\begin{bmatrix} & & 1 & & \\ 1 & & & 1 & \\ & 1/2 & & 1/2 & \\ & & & 1/2 & \\ & 1/2 & & & \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} s_0^{(0)} \\ s_1^{(0)} \\ s_2^{(0)} \\ s_3^{(0)} \\ s_4^{(0)} \end{bmatrix}}_{\vec{S}^{(0)}} + \alpha \cdot \underbrace{\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\vec{e}}$$

second iteration: $\vec{S}^{(2)} = (1-\alpha)\cdot P \cdot \vec{S}^{(1)} + \alpha\vec{e}$

third iteration: $\vec{S}^{(3)} = (1-\alpha)\cdot P \cdot \vec{S}^{(2)} + \alpha\vec{e}$

$$\cdots$$

Stop when $\left\| S^{(t+1)} - S^{(t)} \right\|_1 < tolerance$



- More topics ⟶ more topic-specefic PR vector
- Lager graph ⟶ Lager PR vector

Online Computation

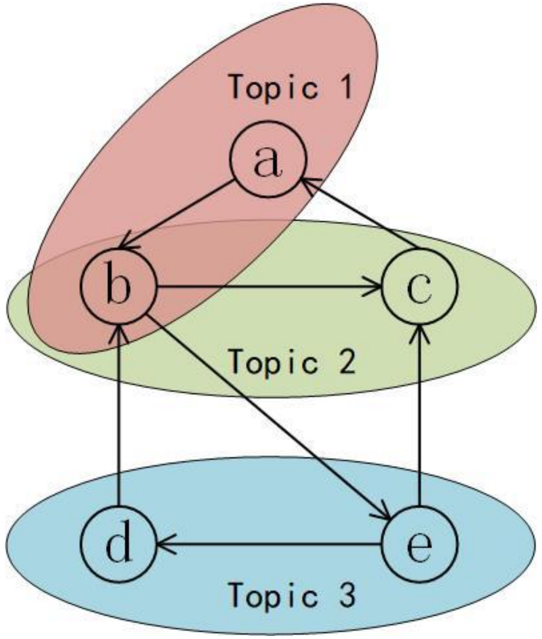
- How to accelerate iterative computation?
  - Approximation: Can't be used in high-precision situations
  - Index: Can't deal with dynamic graph

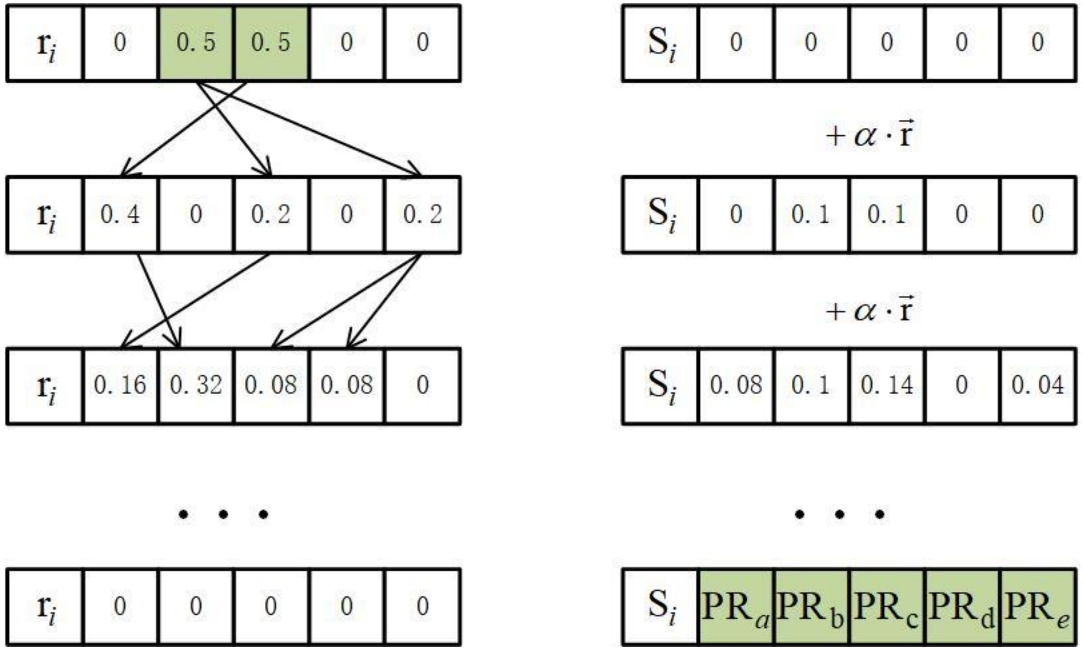
Forward Push

for each vertex :

1. reserve $1-\alpha$ proportion of residual it has received;
2. evenly push the remaining α proportion to its target vertices.

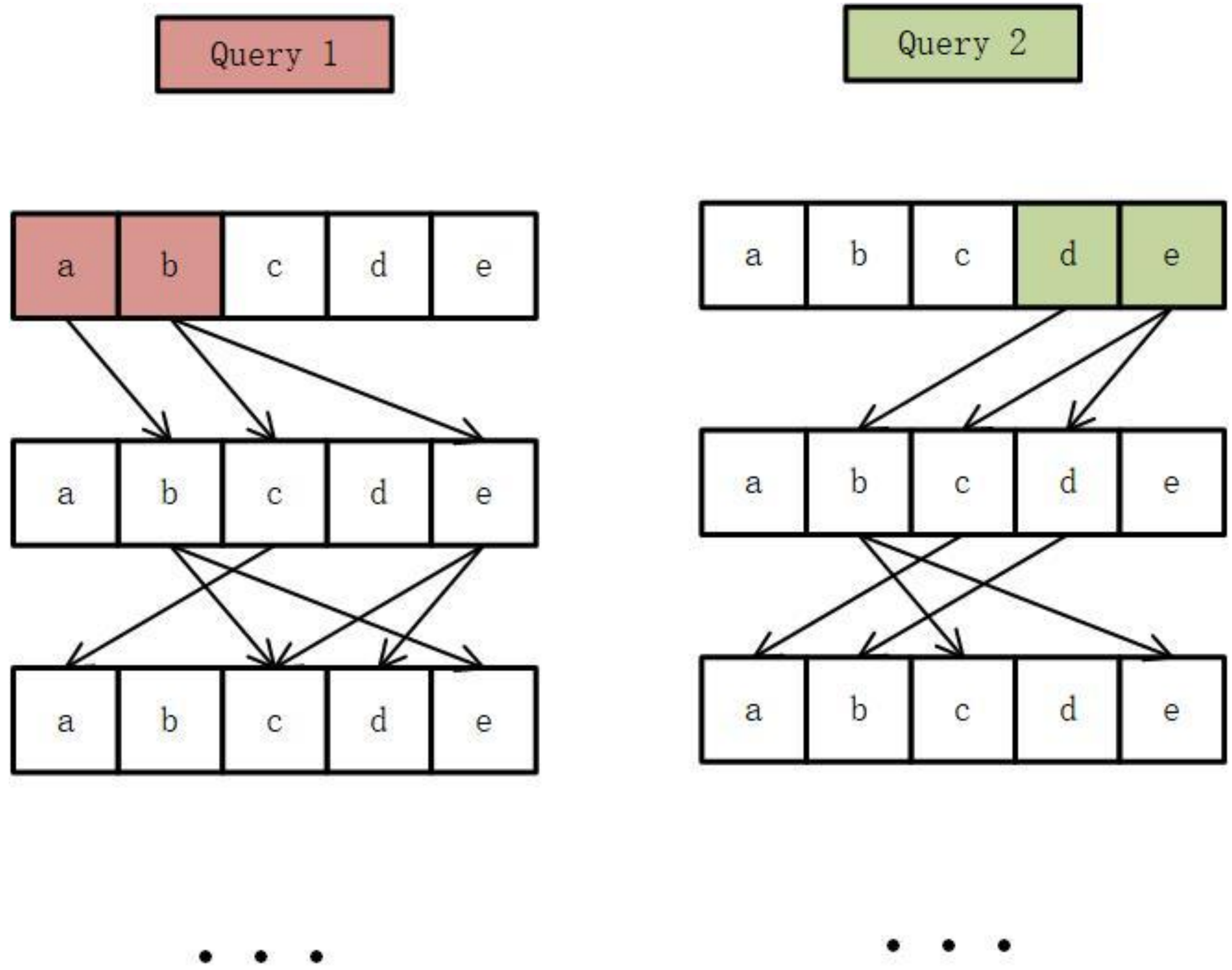
until: no vertex holds valid pushing mass



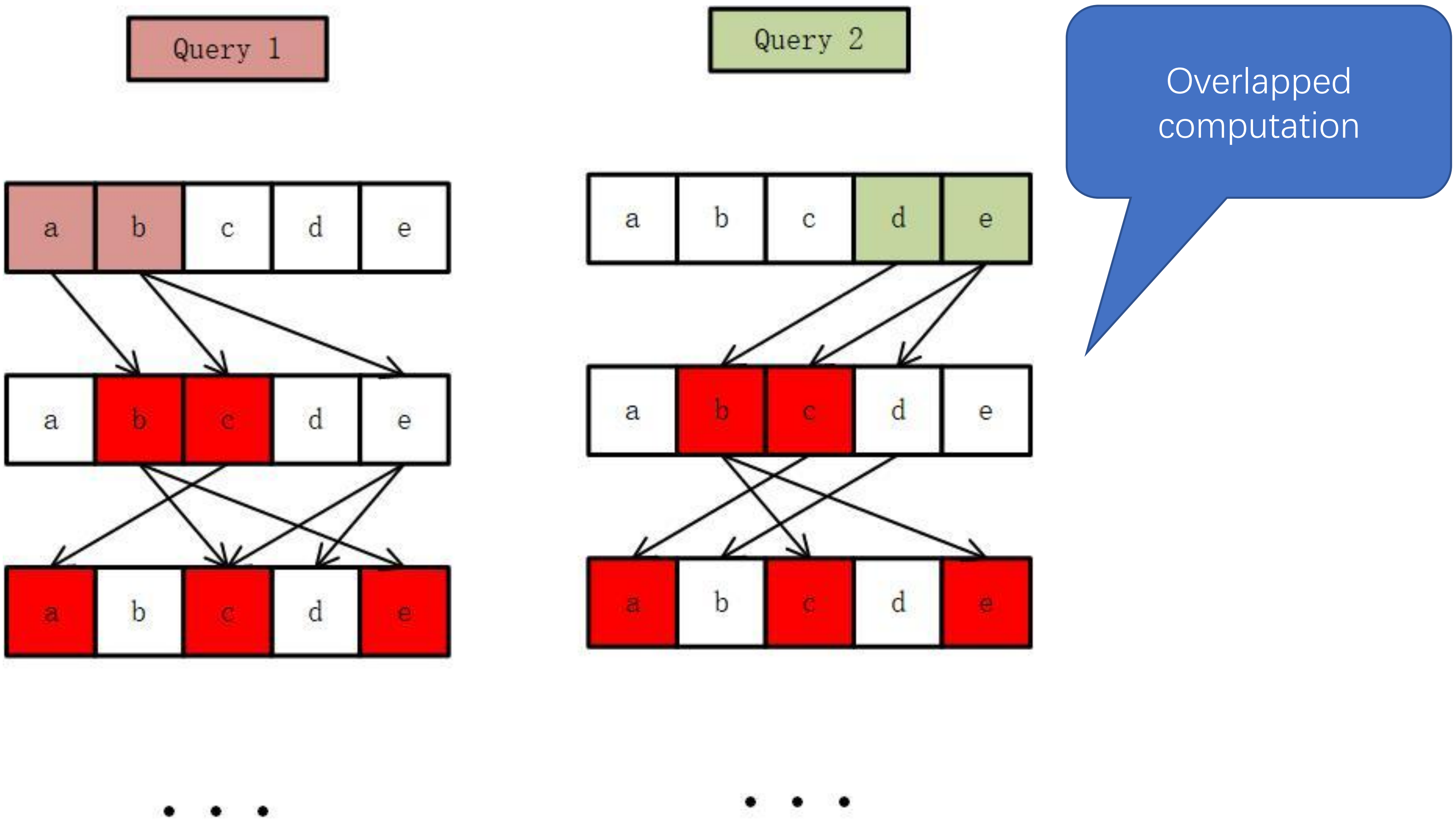


Overlapped Computation

Start from different topic

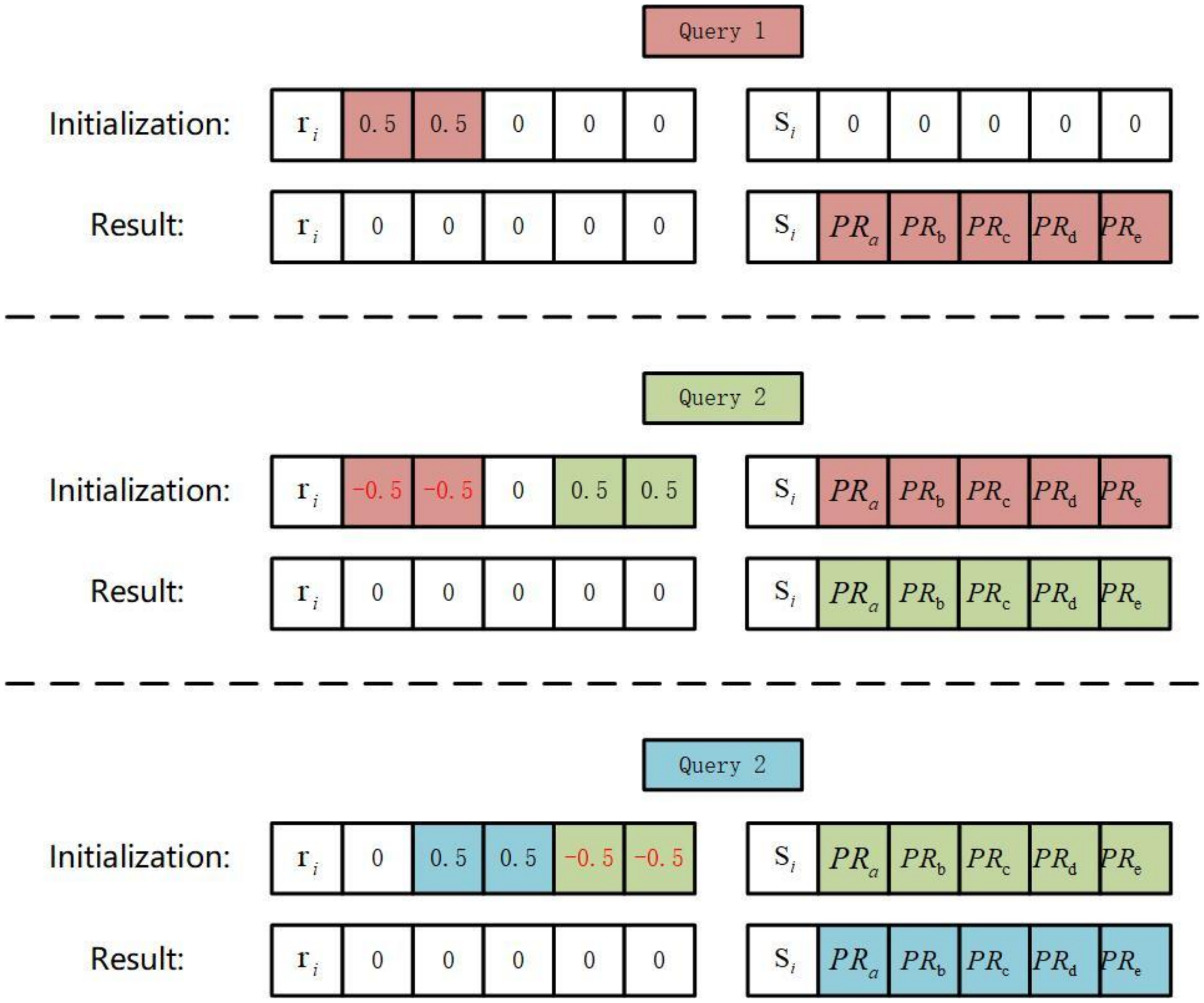


Overlapped Computation



FasTSPR

Core idea: accelerate the current computation by utilzing the results of the previous query

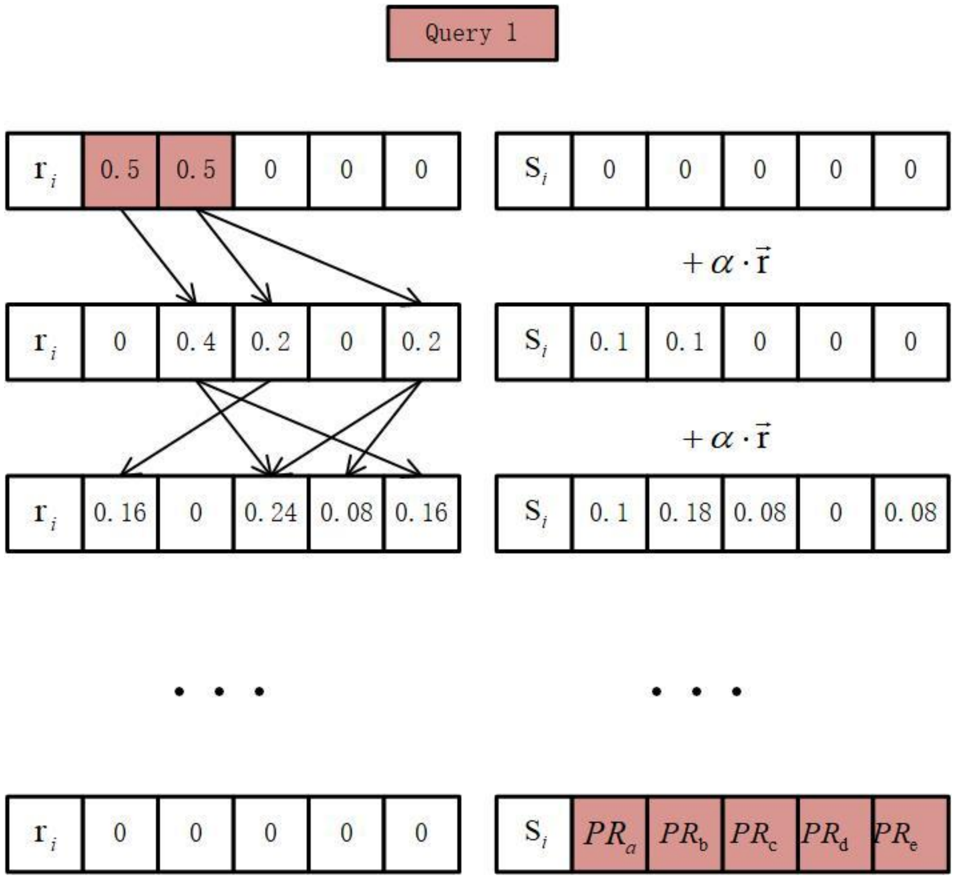


First query: execute the standard forward push method

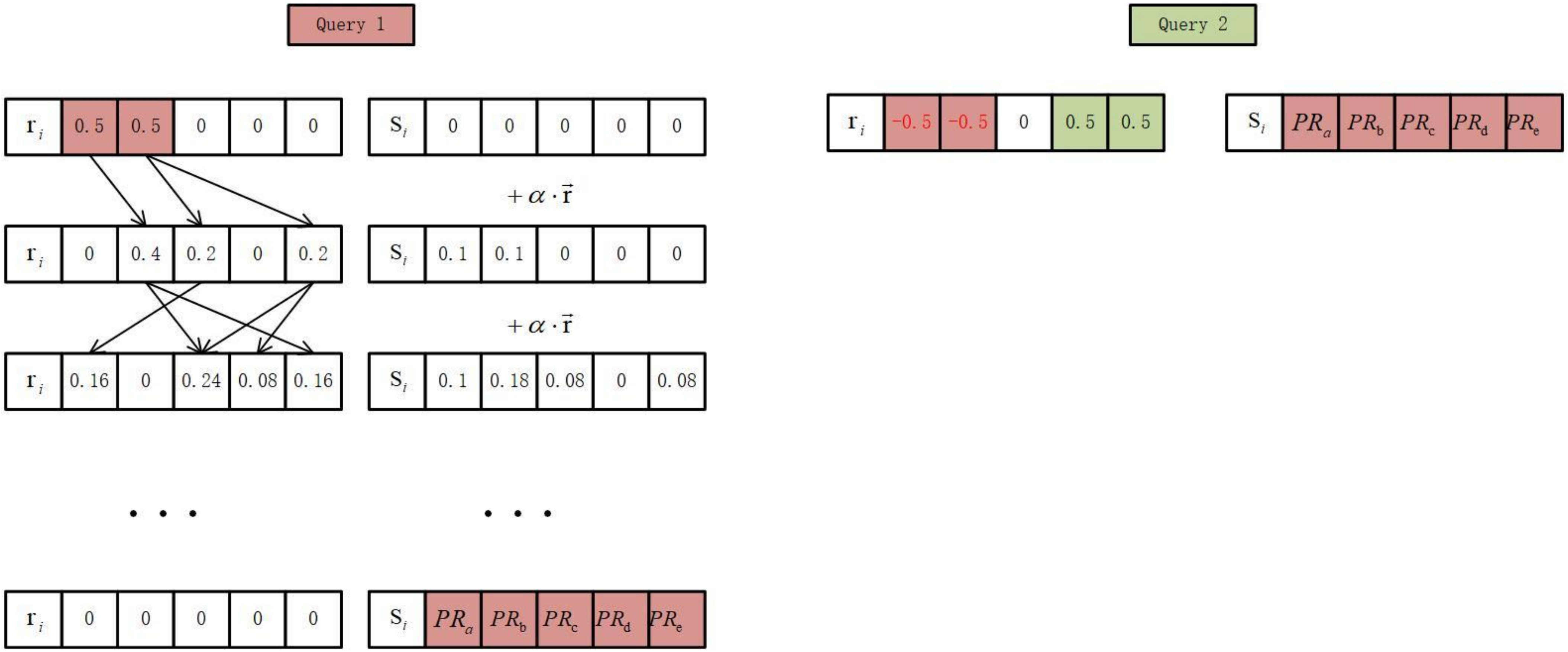
Subsequent query:

1. initialize residual vector: set the current query's topic to a positive value and the previous query's topic to a negative value.
2. initialize socre vector: set score vector to the result from the previous query
3. perform forward push opearation

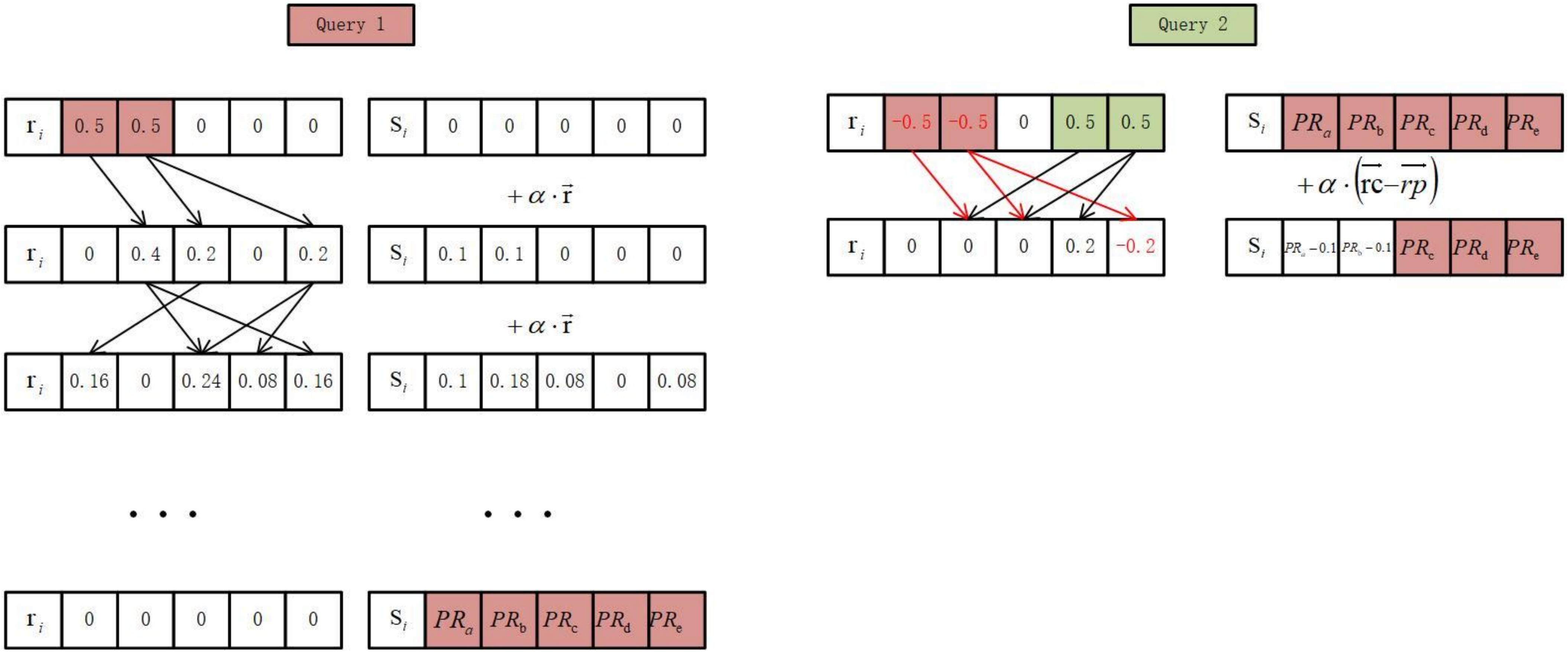
FasTSPR


Query 1
r_i | 0.5 | 0.5 | 0 | 0 | 0
S_i | 0 | 0 | 0 | 0 | 0
$+\alpha\cdot\vec{r}$
r_i | 0 | 0.4 | 0.2 | 0 | 0.2
S_i | 0.1 | 0.1 | 0 | 0 | 0
$+\alpha\cdot\vec{r}$
r_i | 0.16 | 0 | 0.24 | 0.08 | 0.16
S_i | 0.1 | 0.18 | 0.08 | 0 | 0.08
r_i | 0 | 0 | 0 | 0 | 0
S_i | PR_a | PR_b | PR_c | PR_d | PR_e

FasTSPR

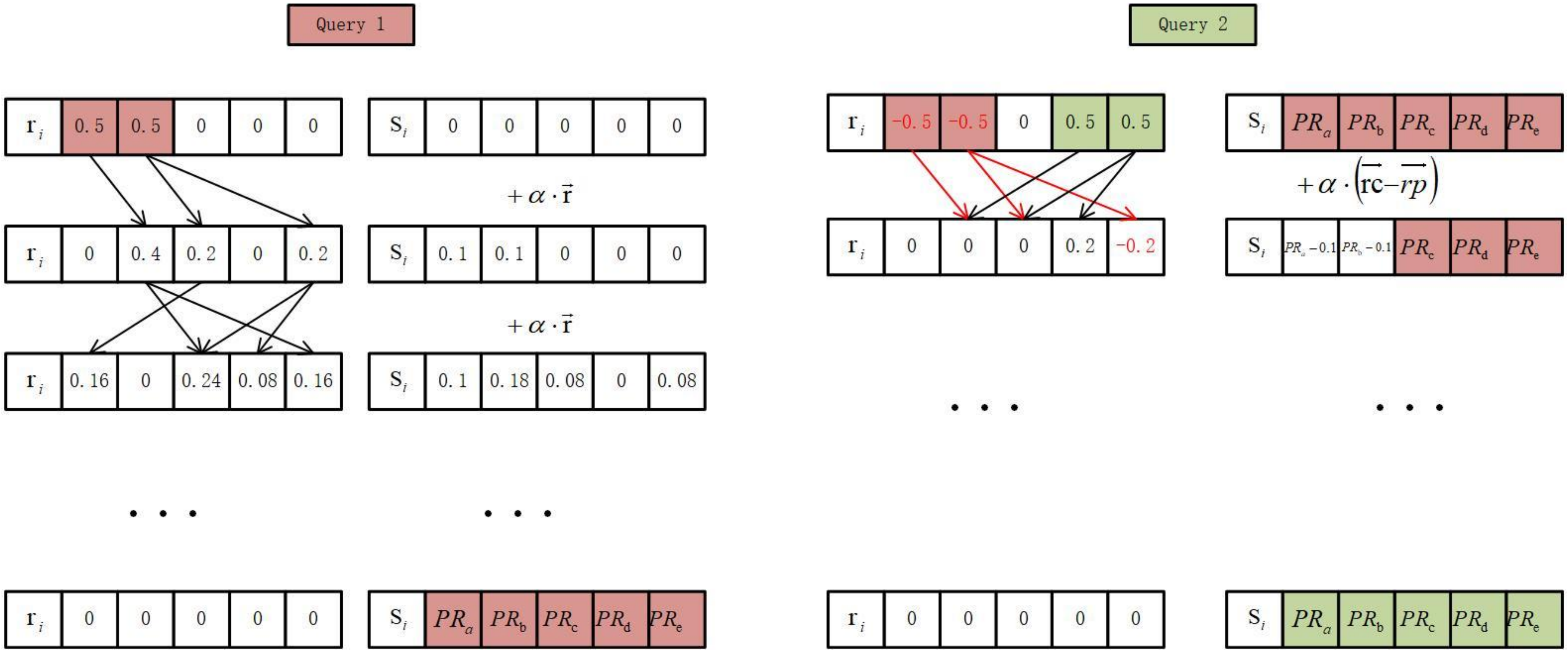

Query 1
Query 2
r_i | 0.5 | 0.5 | 0 | 0 | 0
S_i | 0 | 0 | 0 | 0 | 0
$+\alpha \cdot \vec{r}$
r_i | 0 | 0.4 | 0.2 | 0 | 0.2
S_i | 0.1 | 0.1 | 0 | 0 | 0
$+\alpha \cdot \vec{r}$
r_i | 0.16 | 0 | 0.24 | 0.08 | 0.16
S_i | 0.1 | 0.18 | 0.08 | 0 | 0.08
r_i | 0 | 0 | 0 | 0 | 0
S_i | PR_a | PR_b | PR_c | PR_d | PR_e
r_i | -0.5 | -0.5 | 0 | 0.5 | 0.5
S_i | PR_a | PR_b | PR_c | PR_d | PR_e

FasTSPR



Query 1
Query 2
r_i 0.5 0.5 0 0 0
S_i 0 0 0 0 0
$+\alpha \cdot \vec{r}$
r_i 0 0.4 0.2 0 0.2
S_i 0.1 0.1 0 0 0
$+\alpha \cdot \vec{r}$
r_i 0.16 0 0.24 0.08 0.16
S_i 0.1 0.18 0.08 0 0.08
r_i 0 0 0 0 0
S_i PR_a PR_b PR_c PR_d PR_e
r_i -0.5 -0.5 0 0.5 0.5
S_i PR_a PR_b PR_c PR_d PR_e
$+\alpha \cdot (\vec{rc} - \vec{rp})$
r_i 0 0 0 0.2 -0.2
S_i $PR_a - 0.1$ $PR_b - 0.1$ PR_c PR_d PR_e

FasTSPR



Query 1
Query 2
r_i 0.5 0.5 0 0 0
S_i 0 0 0 0 0
+α·r
r_i 0 0.4 0.2 0 0.2
S_i 0.1 0.1 0 0 0
+α·r
r_i 0.16 0 0.24 0.08 0.16
S_i 0.1 0.18 0.08 0 0.08
r_i 0 0 0 0 0
S_i PR_a PR_b PR_c PR_d PR_e
r_i -0.5 -0.5 0 0.5 0.5
S_i PR_a PR_b PR_c PR_d PR_e
+α·(rc−rp)
r_i 0 0 0 0.2 -0.2
S_i PR_a−0.1 PR_b−0.1 PR_c PR_d PR_e
r_i 0 0 0 0 0
S_i PR_a PR_b PR_c PR_d PR_e

Experiments

- Competitors
  Traditional Power Iteration Forward Push, hybrid method PowerPush
- Environment
  Linux server, 32 GB RAM, Ubuntu 22.04 (64-bit), GCC 11.3
- Datasets

Graph	# of vertex	# of edge
Catster (CT)	149684	10896394
DBLP (DL)	317080	2099732
Google (GL)	916428	12156500
Wiki-Talk (WT)	2394385	8505513
cit-Patents (CP)	3774768	16518948

Overall Performance

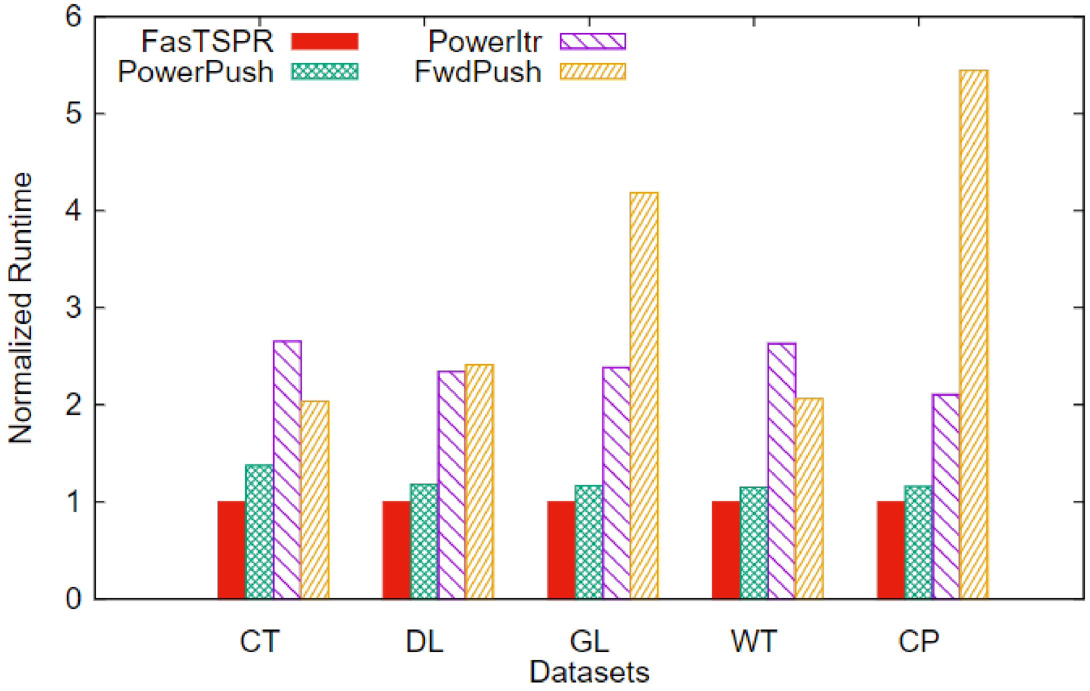


Fig. 1: The comparison of runtime

Achieved a 1.37× - 5.44× speedup

Convergence Speed

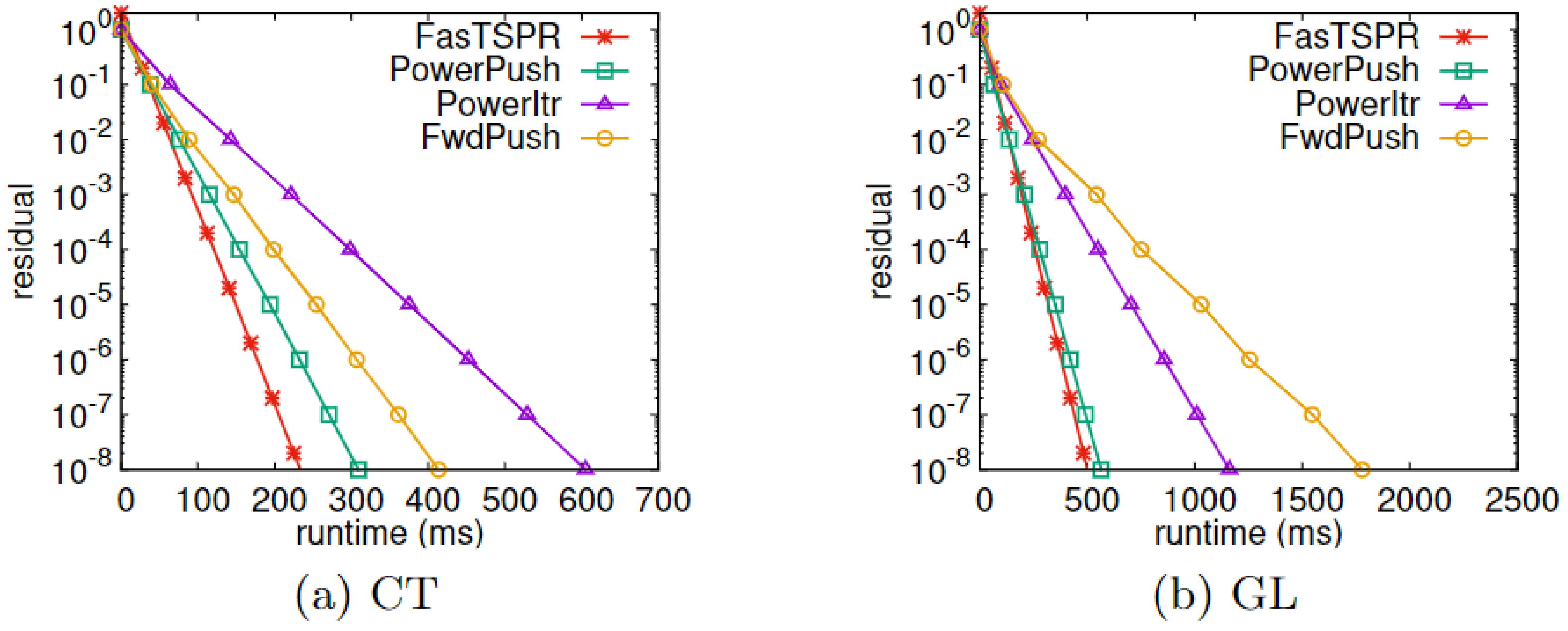
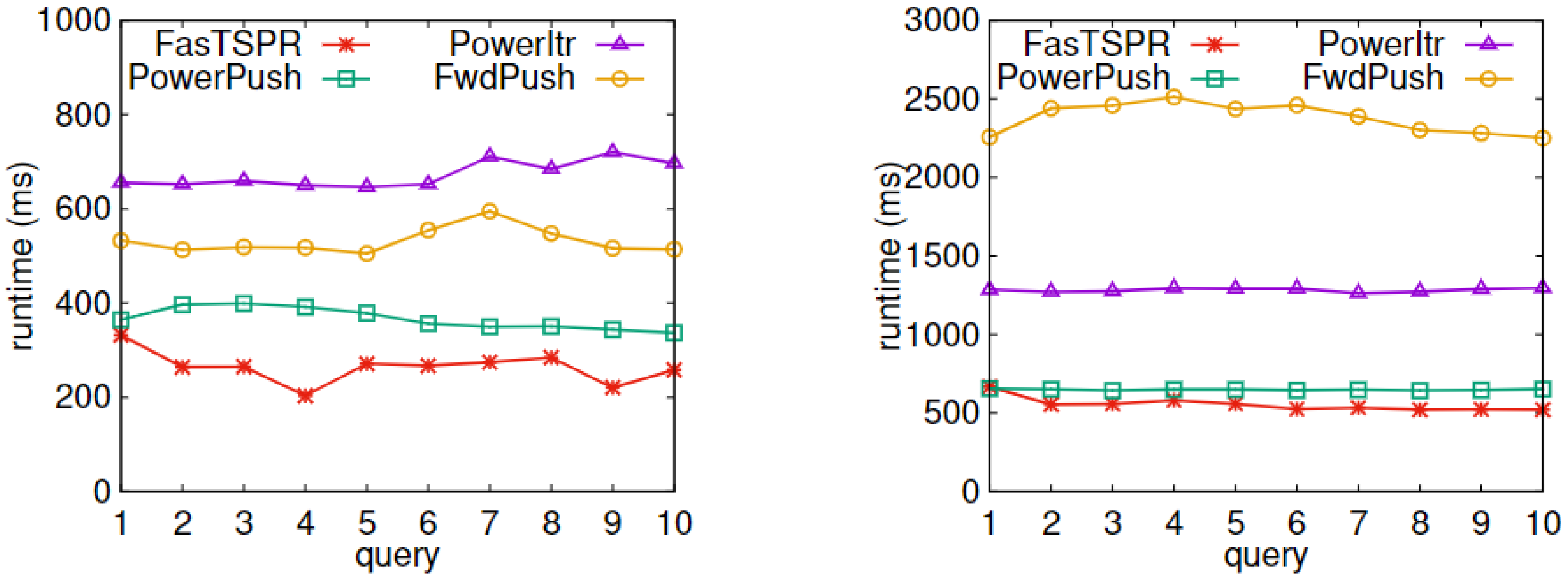


Fig. 2: The comparison of convergence speed

FasTSPR converges the fastest

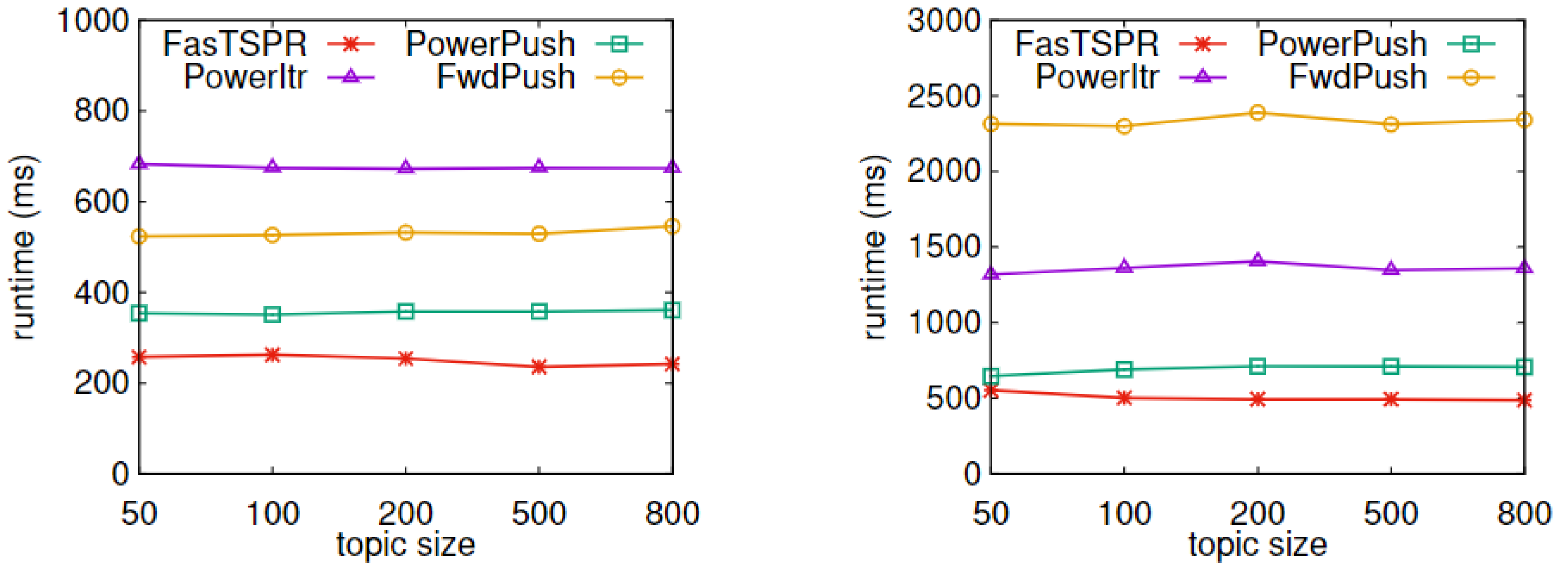
Impact of Different Queries



(a) CT (b) GL

Fig. 3: The runtime when varying the topics

Impact of topic size



(a) CT (b) GL

Fig. 4: The runtime when varying the size of topics

FasTSPR outperforms other algorithm in all query size

Conclusion

- We propose an efficient TSPR algorithm, FasTSPR
- We provide a formal proof to prove the correctness of FasTSPR

Thank you for listening!