

Propagating Maximum Capacities for Recommendation

Ahcène Boubekki^{1,2} and Ulf Brefeld¹ and Cláudio L. Lucchesi³ and Wolfgang Stille⁴

¹ Leuphana University of Lüneburg, Germany {boubekki, brefeld}@leuphana.de

² German Institute for Educational Research, Germany boubekki@dipf.de

³ Universidade Federal de Mato Grosso do Sul, Brazil lucchesi@gmail.com

⁴ Technische Universität Darmstadt, Germany stille@ulb.tu-darmstadt.de

Abstract. Neighborhood-based approaches often fail in sparse scenarios; a direct implication for recommender systems exploiting co-occurring items is often an inappropriately poor performance. As a remedy, we propose to propagate information (e.g., similarities) across the item graph to leverage sparse data. Instead of processing only directly connected items (e.g. co-occurrences), the similarity of two items is defined as the maximum capacity path interconnecting them. Our approach resembles a generalization of neighborhood-based methods that are obtained as special cases when restricting path lengths to one. We present two efficient online computation schemes and report on empirical results.

Keywords: recommender systems, information propagation, maximum capacity paths, co-occurrence, sparsity, cold-start problem

1 Introduction

Recommender systems often utilize co-occurrences of items to pass information between users and items. The underlying idea is that users sharing many items are considered similar and that an item is likely being recommended when it frequently co-occurs with an actually viewed one. Many systems use explicit co-occurrences of items to compute recommendations [28, 16, 13, 19], but also collaborative methods such as neighborhood-based approaches [15, 23] and matrix factorisation techniques [21, 5, 12] inherently ground on leveraging co-occurrence data.

However, the glory of collaborative filtering is quickly turned into a major limitation in the presence of sparsity; the cold-start problem being only one extreme case of this observation. If two items never explicitly co-occur in the data, they cannot be utilized by the recommendation engine at all. In large systems, co-occurrence matrices are naturally large-scale, but sparse. The sparsity however does not change much over time due to the fact that the size of the matrix grows quadratically in the steadily increasing number of items, but the number of matrix entries increases only linearly in the number of co-occurring items. In practice, block structures are frequently observed. This implies several strongly connected components (e.g., genres, subjects, categories) that are again loosely connected to each other.

The cold-start problem appears in many situations, e.g., at the very beginning of collecting data, or when adding new items or new users. In the first case, not enough

information has yet been collected to distinguish items. In the latter cases, the new items and users are isolated and connect to the others with only a few edges. In these scenarios, user tendencies cannot be inferred with high confidence; hence, co-occurrence-based methods are prone to fail and new techniques are required to extend neighborhoods of items appropriately.

In this paper, we study recommender systems for sparse settings to tackle cold-start problems and data sparsity. To increase the item neighborhood, we propose to leverage the transitive hull of all maximum capacity paths in the item graph, such that local item neighborhoods are extended to other connected components of the item graph. By doing so, co-occurrences are propagated through the item graph and similarities between all items, that are connected by at least one path, can be computed. We present two efficient online algorithms for integral as well as arbitrary capacities, respectively. We empirically evaluate our approaches on synthetic and standard datasets. In controlled cold-start scenarios, we identify limitations of collaborative filtering-based methods and show that the proposed approaches lead to better and more precise results.

The remainder is structured as follows. Section 2 reviews related work and Section 3 introduces the problem setting. We present our main contribution in Section 4. Section 5 reports on our empirical results and Section 6 concludes.

2 Related Work

Neighborhood-based collaborative filtering (CF) methods are perhaps the most widely used type of recommender systems. Although user-based approaches are not practical in real settings because of scalability issues [23], item-based collaborative filtering is very common. Besides being conceptually simple, a major advantage of CF-based approaches is the similarity measure that can be adapted to a problem at-hand. Common choices are for instance co-occurrence, cosine, correlation, Pearson, as well as hybrid similarity measures [1].

Co-occurrence data is frequently the basis for recommender systems [28, 16] and approaches range from co-occurrence data sampled from random walks in item-item graphs [13] to generalizations of the aspect model [10] by considering three-way co-occurrences between users, items and item content [19]. The resulting generative models may also address cold-start problems [24].

Intrinsically, matrix factorization approaches [14, 22] are neighborhood-based algorithms as well; high dimensional vectors are embedded in compact spaces by low dimensional (linear) transformations while preserving the notion of similarity. This class of algorithms performs well when the amount of training data is sufficiently large, but causes problems when data is sparse. Examples for such sparse scenarios are again cold-start and cross-selling problems [27]. With respect to collaborative filtering, this problem is handled by using additional information on the user like age, gender, etc. [4]. Another approach is to consider transitivity closures of item-graphs [2]. An application to recommender system is for example proposed in [25] using a Jaccard similarity measure.

To address recommendation in sparse scenarios, we rephrase the task as a propagation problem in an item-graph [6]. Drawing from network flow theory, we

focus on the maximum capacity path [18, 11] between two items. In general, using the maximum flow model would be appropriate as well, but as graphs in our application are usually very large, but sparse, the additional complexity spent on flow algorithms does not seem to be adequate with respect to the fact that in sparse scenarios, there is only one path at all, if a connection between items exists at all.

Maximum capacities have been used together with recommender systems by Malucelli et al. [17] before. The authors define similarity as the solution of a bi-criterion path problem [9] that is computed using a Dijkstra-based algorithm. Unfortunately, the authors choose some cumbersome definitions instead of linking their contribution to graph theory, and do not investigate the effect of sparsity. Unfortunately, the proposed algorithm has a complexity of $\mathcal{O}(n^4)$, thus rendering its application infeasible in large-scale scenarios. Nevertheless, it is a useful approach for small-scale problems. We review their contribution in Section 3.3 and use the algorithm as a baseline in Section 5.

3 Preliminaries

3.1 Rationale

The rationale behind our approach is as follows. Considering weighted graphs, we require a function that translates the transitive closure in the binary case to weighted problems. Shortest path trees are such a translation in case of an additive weight function. However, we do not have additive weights in our case, otherwise the applied measure (e.g. co-occurrence) would increase with the number of edges on a path, which is counterintuitive.

There are approaches normalizing shortest paths by the number of hops they consist of, i.e. computing the average weight of a path. This is not reasonable either as paths consisting of many strongly weighted edges and just one lowly weighted (bridging) edge, have a strong average weight despite of the fact that they virtually decompose into two connected components. Multiplicative approaches are not appropriate either as the weight of the path might be lower than the lowest weight of one of its edges, in particular when it is normalized by the number of hops.

Therefore, we focus on the minimum weight of the path's edges for correlating two items interconnected by that path. This choice seems appropriate as the weights of transitive connections between two nodes are independent of the length of the path in the first place. Bottlenecks are addressed in a reasonable way: the weight of the transitive connection is determined by the lowest weighted edge it contains. Eventually, for two nodes u and v , we consider the connecting path that has the highest minimum edge weight among all other connection path, i.e. the *maximum capacity path*.

3.2 Maximum Capacities

Let $G = (V, E)$ be a weighted graph, where V is a set of n nodes and E a set of m edges. Furthermore, let $A = (a_{ij})$ be its related adjacency matrix. In order to ease the following definitions, we assume without loss of generality that G is acyclic, and all

self similarities are set to infinity (i.e. $a_{ii} = \infty$)[†]. The capacity of a path is defined as follows.

Definition 1 (Capacity).

Let $p = ((u, u_1), \dots, (u_{l-1}, v))$ be a path between two vertices $u = u_0$ and $v = u_l$. The capacity, $c(p)$, of p is defined as the minimum weight of its edges :

$$c(p) := \min_{(u_k, u_{k+1}) \in p} (a_{u_k u_{k+1}}).$$

Since there may be more than just one connecting path for a pair of nodes, we regard the path that implies the maximum capacity among all paths.

Definition 2 (Maximum Capacity).

Let $\mathcal{P}_{uv} = \{p_0, \dots, p_k\}$ be the set of all the paths between u and v . The maximum capacity, $mc(u, v)$, between u and v is defined as maximum of all the paths capacities in \mathcal{P}_{uv} :

$$mc(u, v) := \max_{p \in \mathcal{P}_{uv}} (c(p)).$$

Computing maximum capacity paths with a minimum number of hops using max-min matrix products is prohibitively expensive with a complexity of $\mathcal{O}(n^4)$. The Dijkstra algorithm is able to efficiently compute maximum capacity paths from a dedicated start node in $\mathcal{O}(n \log(n) + m)$. This amounts to $\mathcal{O}(n^2 \log(n) + mn)$ when considering shortest paths between all pairs of nodes. Before we introduce two efficient algorithms in Section 4, we briefly review the bi-criterion shortest path approach by [17].

3.3 Bi-criterion Shortest Path

Malucelli et al. [17] propose a capacity-based algorithm for recommendation. Their algorithm also considers the transitive closure of the graph, however the edge weights are given by solutions of a bi-criterion shortest path optimization problems, as shown in the following definition.

Definition 3 (Bi-criterion Capacity).

Let $\mathcal{P}_{uv} = \{p_0, \dots, p_k\}$ be the set of all the paths between u and v . The length of a path p is denoted as $\lambda(p)$ and its capacity as $c(p)$. The bi-criterion capacity, $bc(u, v)$, between u and v is defined as follows:

$$bc(u, v) := \max_{c(p)} \left(\frac{c(p)}{\lambda(p)} \right).$$

One problem with this approach is the fact that the same result (maximum capacity paths with a minimum number of hops) might be computed by a much simpler algorithm: (i) compute (all) maximum capacity paths (e.g. by using Dijkstra), and (ii) finally choose the one with the minimum number of hops. This can be achieved in

[†] Note that paths are usually cycle free by definition and capacities do not change by repeating cycles.

$\mathcal{O}(n^3)$ in total, whereas the approach proposed in [17] takes $\mathcal{O}(n^4)$ and is much more complex to implement.

Note that similar problems have been studied in [7]. In this paper, we use the bi-objective label correcting algorithm with node-selection introduced by [26]. See [20] for an extensive review.

4 Maximum Capacity Algorithms

The following two sections are dedicated to two online algorithms that efficiently compute maximum capacity paths. Section 4.1 approaches the problem by a bucket style approach in case all weights are integral. Section 4.2 uses a tree-based approach and works with arbitrary edge weights.

4.1 Max Capacity Buckets

The basic idea is to consider the sub-graphs G_α of G , referred to as *buckets*, containing all the edges with weights higher than a given $\alpha \in \mathbb{N}_0$. In theory, α might be a real number, however in practice it can only be an integer as each of them needs to be stored separately. Note that $G_0 = G$. As the graphs G_α are acyclic and not necessarily connected, they can also be seen as forests.

$$G_\alpha = (V_\alpha, E_\alpha) \text{ such that } \begin{cases} E_\alpha := \{(i, j) \in E : a_{ij} > \alpha\} \\ V_\alpha := \{i \in V, j \in V : (i, j) \in E_\alpha\}. \end{cases}$$

The update of the buckets is straightforward. When the user views a new item v , edge to previously seen item (u, v) are updated as follows:

- the adjacency of the undirected edge is increased by one : $a_{uv} = a_{uv} + 1$,
- if not in $V_{a_{uv}}$, the nodes u and v are added to $G_{a_{uv}}$ as trivial trees,
- finally, the connected components containing u and v are merged: the root of the smallest tree becomes a child of the other root.

The maximum capacity between two nodes u and v is given by the biggest index of the buckets containing u and v .

$$mc(u, v) := \max(\alpha : (u, v) \in V_\alpha)$$

Graphs are implemented as dictionaries to leverage speed of their key search. The first step is just a modification or a creation of keys in a dictionary that is $\mathcal{O}(1)$ in average. The next step check if nodes are in a subgraph. In case of a positive answer this is done in constant time. However as weights grow, it will occur that at least one of the nodes will return a negative answer. The test will then have gone through all the keys at a linear cost of $\mathcal{O}(n)$. The final step needs to find the roots of two trees which can be done in $\mathcal{O}(\log n)$. Merging is performed by altering the property of the nodes which is constant $\mathcal{O}(1)$. In total, an update has a complexity of $\mathcal{O}(n + \log(n))$.

The evaluation of the maximum capacity between two nodes is done by browsing the buckets in increasing weight order until one of the nodes is not included anymore.

Again, the respective test in each sub-graph requires in average $\mathcal{O}(1)$. The search stops when a test is negative, in which case we have costs $\mathcal{O}(n)$. Hence the overall complexity of the maximum capacity is $\mathcal{O}(a^* + n)$, where a^* is the largest weight given by $a^* := \max_{(u,v) \in E} (a_{uv})$. Note that the computation could be sped up by starting the search from $G_{a_{uv}}$ as the maximum capacity cannot be smaller than the weight of the direct edge, if it exists.

4.2 Max Capacity Trees

The well-known and efficient label correcting algorithm, known as *Dijkstra's Algorithm* [3], for the computation of shortest paths in directed graphs from a source node s might be easily adapted for the computation of maximum capacity paths. The distances/maximum capacities need to be initialized with $-\infty$ instead of ∞ . Paths are then built using the following alternative update scheme preserving time and space complexity, where π is the predecessor function :

```

for each neighbor  $v$  of  $u$ :
  if  $mc(s, v) > \min\{mc(s, u); a_{uv}\}$  :
     $mc(s, v) := \min\{mc(s, u); a_{uv}\}$ 
     $\pi(v) := u$ 
  end if

```

The algorithm yields a tree rooted at some specified start node s containing all nodes of the item graph, each with a label determining the capacity of the unique (maximum capacity) path to the root node s . The algorithm is computed several times such that each node of the graph becomes the start node exactly once. We obtain n rooted maximum capacity trees. The runtime of Dijkstra amounts to $\mathcal{O}(m + n \log n)$, where $n = |V|$, and $m = |E|$, so this yields to $\mathcal{O}(nm + n^2 \log n)$ for the all-pairs variant.

Runtime complexity might be reduced to $\mathcal{O}(n^3)$ in total by using an *all-pairs shortest path algorithm*, for example the *Floyd-Warshall* algorithm, which makes use of two $n \times n$ matrices: a capacity matrix MC stating the maximum capacities $mc(u, v)$ for each path between a pair of nodes u and v , and a matrix Π holding the predecessor labels $\pi_{u,v}$ for the reconstruction of a max capacity path connecting u and v . This is beneficial in case of an online algorithm, where edges are generated successively, as updates take place in two matrices instead of n different tree structures, that contain specific edges weight.

Despite a concrete implementation as single source or all-pairs algorithm, we use the label correction idea to develop an online algorithm for the computation of maximum capacity trees. The algorithm maintains a forest at any time. As with shortest path trees, the trees contain maximum capacity paths. Initially, we start with a forest of trivial trees consisting of only a single node each. There are no edges and weights are strictly nonnegative, hence $mc(u, v) = 0$ for all $(u, v) \in V^2$. We add edges (co-occurrences) successively. Whenever an edge (u, v) with weight w is added, the following cases may occur:

- (i) there is already a tree edge (u, v) with $mc(u, v) < w$: set $mc(u, v) := w$,

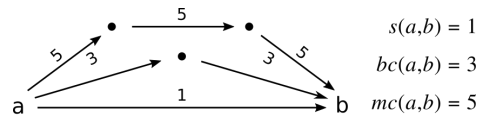
- (ii) there is no tree edge (u, v) yet, u and v are part of different trees: connect the two trees by inserting (u, v) and set $mc(u, v) := w$,
- (iii) there is no tree edge (u, v) yet, u and v are part of the same tree: inserting (u, v) would generate a cycle. In order to avoid that, follow the path from u to v and determine the lowest capacity weight edge(s) on this path: if w is lower, leave everything as is, otherwise remove the lowest capacity weight edge (in case there are more than one, choose one arbitrarily), and insert (u, v) instead.

Finally, we have to update the matrices according to the above operation: update predecessor labels $\pi_{u,v}$ in case of (ii) and (iii), and update the capacities $mc(u, v)$ in all three cases. The runtime complexity of an update operation is bounded by $\mathcal{O}(n^2)$.

The determination of the item(s) to recommend connected to an item u by paths of maximum capacity amounts to looking up the maximum in one column or row, respectively, of the matrix. This amounts to $\mathcal{O}(n)$. Determining the $n - 1$ recommendations to all other nodes in decreasing order of their connecting maximum capacity paths amounts to sorting that column or row in decreasing order by their capacities. This amounts to $\mathcal{O}(n \log n)$. Note that the tree-based variant can be implemented for arbitrary, i.e. real-valued edge capacities.

4.3 Discussion

Using bi-criterion capacities as in [17] favors shorter paths; hence closer items are ranked higher. However, all items in the connected components are still reachable. In that sense, the bi-criterion approach ranges in between purely neighborhood-based and our maximum capacity-based approaches. Consider the following example:



A simple neighborhood-based approach will simply connect the nodes a and b by their connecting edge and leads to a score of one. The bi-criterion algorithm however enumerates all paths between a and b and decides for the aggregated score of three as the capacity is weighted by the length of the path. Finally, maximum capacity also enumerates all paths from a to b but chooses the maximum of the three capacities as the resulting score.

Note that the two maximum capacity-based algorithms described in Section 4 tackle the problem differently, but produce the same results. Due to lack of space, we leave this textual statement without formal proof as the algorithmic invariants are easy to verify. In the empirical evaluation, we compare both in terms of their complexities but focus on the tree-based approach when we study performance.

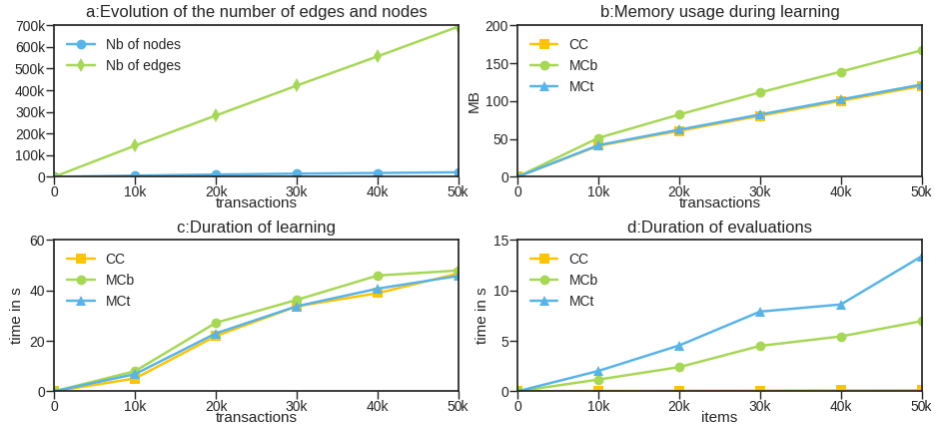


Fig. 1: Runtime comparisons

5 Empirical Evaluation

5.1 Time and Space Complexity

To compare the complexities of the proposed algorithms in time and space, we train the buckets (MCb) and the tree-based (MCT) algorithms on a synthetic dataset made of 21k items and 50k transactions. The experiment consists of sequentially reading and processing/storing the data. Except for the first transaction, the maximum capacity of each transaction is computed on its arrival to update the buckets or trees. For comparison, we include as a baseline the cost of only building and reading the adjacency matrix A .

Figure 1 shows the results for varying training set sizes. The upper left sub-figure shows the linear growth of edges in the size of the data. The bottom left sub-figure indicates that all three algorithms learn at the similar pace. This result is not surprising as all three algorithms have to learn the adjacency matrix. However, it is interesting to see that updating the buckets or the trees is negligible compared to only adjacencies.

On the other hand, the bottom right sub-figure shows that the data structure is very important for the capacities. The co-occurrence baseline is extremely fast as it simply retrieves coefficients from the matrix. By contrast, the tree structure proves very costly when it comes to retrieve elements. The tree structure has to build the path to the root at least once, even if the two nodes are in the same connected component (or on the same tree). This may be overcome by using a matrix holding all maximum capacities between all pairs of nodes such as in the Floyd-Warshall algorithm. Then, evaluations take constant time whereas the complexity of updates remains unchanged. As capacities are monotonously increasing, two $n \times n$ matrices are sufficient, one for the maximum capacities between each pair of nodes, and one for the predecessors.

A drawback of the buckets-based approach is the amount of buckets. There are as many of them as there are different weights. The consequence for the memory usage can be observed in the upper right sub-figure. It should be noted that for both maximum capacity algorithms, the adjacency matrix is built along the buckets or trees. However,

it is not necessary to compute the capacities themselves. In a production system, the adjacency matrix can be discarded once the learning has been done. Hence, the actual memory usage for a recommender system that is based on one of the two propagation algorithms would be the difference of the respective curve and the baseline. This implies that the tree-based method is in fact very efficient.

5.2 Precision and incertitude on cold-start

We compare co-occurrence (CC), cosine (CS), the Bi-Criterion Shortest Path (BCSP) approach by [17] and two variants of maximum capacity (MC) and (MC+dist). The latter aims at breaking similarity blocks by sorting items in a block relatively to the average distance of the maximum capacity paths. Note that as we use the tree-based algorithm, these distances are side products.

We evaluate the different approaches on the MovieLens 1M dataset [8]. It contains over 1 million ratings of 3952 movies made by 6040 users. A rating is defined by a userID, a movieID, a score and a timestamp. The ratings are binarized by thresholding scores greater than three (positive examples). Scores smaller or equal than three are considered negative examples.

The cold-start problem is simulated by reading and processing the data chronologically. If the actual rating is for a movie rated less than 20 times more but at least once, a recommendation is computed using the average similarity to the previously positively rated items of the user. If it is the user's or the item's first rating, it is skipped. The threshold of 20 is chosen as we are interested in the early stage of the cold start problem. For fairness, only the first thousand recommendations are reported, as BCSP requires more than a day to complete the experience. At this point the adjacency matrix reaches approximately 1% of sparsity. The experience is repeated six times by skipping respectively none, the 2k, 4k, 6k, 8k and 10k first ratings. The reported results are the average of the six iterations.

A limitation appears as collaborative filtering cannot distinguish the movies that have the same similarity. As a consequence, the returned ranking will be made of blocks of the same values. The size of the block containing the expected item is called *incertitude*. Note that this situation is frequent in this early stage of a recommender system. In our evaluation, we assume that the item is ranked in the middle of the block of items sharing the same similarity. This amounts to treating items as randomly distributed inside each block.

Figure 2 shows the evolution of four standard metrics: Success@10 defined as the number of times the desired movie has been ranked in the top 10 for the first thousand recommendations, Precision@10, Recall@10 and Mean Reciprocal Rank. For clarity the standard deviation of the measurement are shown only for CC and MC+dist represented by the shaded area around the average lines. All curves are smoothed by showing only one point for every 30 recommendations.

The four plots are consistent. Irrespectively of the performance metric, MC+dist performs best. The single ranked version, MC, performs as good as CC but its performance drops fast and starts to follow BCSP while still being better. With increasing recommendations, CC catches up and stays close to MC+dist on average, which is also indicated by the overlapping standard deviations. It is interesting to point

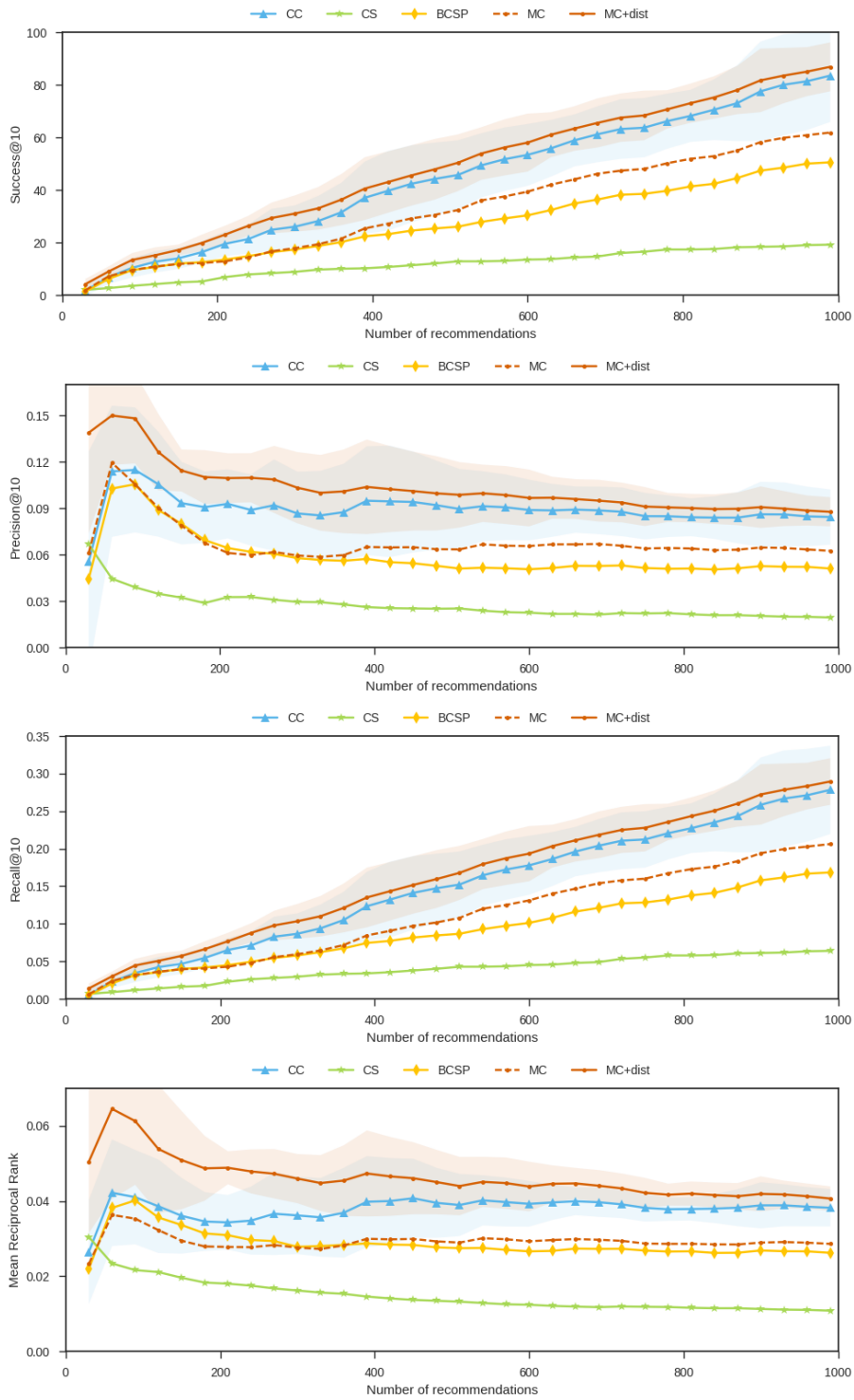


Fig. 2: Standard metrics.

out that MC+dist’s results are more stable over the six repetitions which is shown by small standard deviation compared to that of CC. Not shown in the figure is the break-even point where the two lines decouple and CC becomes more successful on average. However the difference becomes only significant much later.

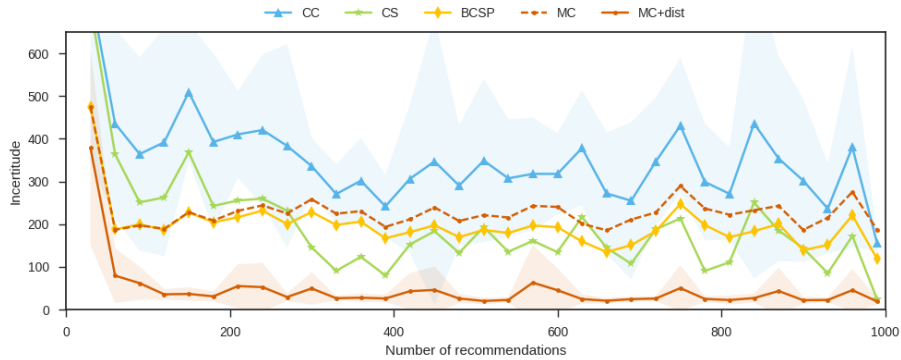


Fig. 3: Evolution of the incertitude of each model’s recommendation.

The superiority of the double ranked MC-based approach is supported by its very low incertitude as shown in Figure 3. Here again, the standard deviation is shown only for CC and MC+dist. Despite its high likelihood to rank the expected item in the top 10, CC is much more *uncertain* than MC+dist. The block of the expected item is on average made of 315 items, while the average for MC+dist is only 39. It is interesting to observe that the double ranking of MC+dist render its performance more accurate than its simpler peer, in terms of precision and ambiguity of the recommendation. Note that this improvement comes almost for free. The MC+dist is also particularly more consistent over the six folds of the experiment, as the standard deviation is again much smaller than that of CC.

6 Conclusion

We cast the problem of recommendation as a propagation problem in an item graph using network flow theory and proposed two efficient online algorithms, a bucket and tree-based one. Empirically, the two algorithms effectively leverage sparse data at enterprise level scales. Our approaches distinguish themselves in cold-start scenarios, in term of precisions and quality of their recommendations. Future work will study the relation with T-transitive closures.

Acknowledgments

This research has been funded in parts by the German Federal Ministry of Education and Science BMBF under grant QQM/01LSA1503C and the Brazilian CAPES Foundations and a grant from CNPq.

References

1. F. Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 273–280. ACM, 2013.
2. B. De Baets and H. De Meyer. On the existence and construction of t-transitive closures. *Information Sciences*, 152:167–179, 2003.
3. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
4. Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the 2010 IEEE International Conference on Data Mining, ICDM '10*, pages 176–185, Washington, DC, USA, 2010. IEEE Computer Society.
5. R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.
6. M. Gori, A. Pucci, V. Roma, and I. Siena. Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*, volume 7, pages 2766–2771, 2007.
7. P. Hansen. *Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979*, chapter Bicriterion Path Problems, pages 109–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980.
8. F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.
9. M. I. Henig. The shortest path problem with two objective functions. *European Journal of Operational Research*, 25(2):281–291, 1986.
10. T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of Uncertainty in Artificial Intelligence*, 1999.
11. T. C. Hu. Letter to the editor—the maximum capacity route problem. *Operations Research*, 9(6):898–900, 1961.
12. Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining*, 2008.
13. U. Kang, M. Bilenko, D. Zhou, and C. Faloutsos. Axiomatic analysis of co-occurrence similarity functions. Technical Report CMU-CS-12-102, School of Computer Science, Carnegie Mellon University, 2012.
14. Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.
15. Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1, 2010.
16. M. Li, B. Dias, W. El-Dereby, and P. J. G. Lisboa. A probabilistic model for item-based recommender systems. In *Proceedings of the ACM Conference on Recommender Systems*, 2007.
17. F. Malucelli, P. Cremonesi, and B. Rostami. An application of bicriterion shortest paths to collaborative filtering. In *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2012.
18. M. Pollack. Letter to the editor—the maximum capacity through a network. *Operations Research*, 8(5):733–736, 1960.
19. A. Popescul, L. Ungar, D. M. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2001.

20. A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.
21. S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the ACM Conference on Recommender Systems*, 2008.
22. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.
23. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the International World Wide Web Conference*, 2010.
24. A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2002.
25. T. Simas and L. M. Rocha. Semi-metric networks for recommender systems. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 03*, pages 175–179. IEEE Computer Society, 2012.
26. A. J. Skriver and K. A. Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27(6):507–524, 2000.
27. J. Wang, B. Sarwar, and N. Sundaresan. Utilizing related products for post-purchase recommendation in e-commerce. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 329–332. ACM, 2011.
28. C. Wartena, R. Brussee, and M. Wibbels. Using tag co-occurrence for recommendation. In *Proceedings of the International Conference on Intelligent Systems Design and Applications*, 2009.