ValuePick: Towards a Value-Oriented Dual-Goal Recommender System

Leman Akoglu
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
lakoglu@cs.cmu.edu

Christos Faloutsos
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
christos@cs.cmu.edu

***Abstract*—Given a user in a social network, which new friends should we recommend, the dual goal being to achieve user satisfaction *and* good network connectivity? Similarly, which new products are better to recommend to satisfy customers' taste/needs *as well as* increase vendor profit? Typical recommender systems use merely past purchases, product ratings, demographic meta-data, and network 'proximity' to make recommendations. This traditional approach, however, does not take into account the profitability of products to vendors in a customer-product network, or the efficacy of new links in a social network. We argue that it is more appropriate to view the problem of generating recommendations as an optimization problem. In this paper, (a) we propose ValuePick, a framework which incorporates the 'value' of recommendations into the system while still providing accurate recommendations that retain user trust; (b) our method is *parsimonious* (requires only a *single* parameter τ), *flexible* (τ is used to flexibly adjust the level of balance between 'user satisfaction' and 'gain'), and *general* (can be used with any 'value' metric); and finally (c) we examine the problem in the social networks setting, simulate comprehensive experiments to compare our method to several basic heuristics, and show that ValuePick yields higher 'gain' while keeping user satisfaction high.**

***Keywords*-recommender systems; value analytics; optimization; customer-product networks, social networks**

I. Introduction

Due to the rapid growth of electronic commerce, target-based marketing has become a core area of research that has been shown to increase cross-sales, help customers become aware of new products, and add to customer loyalty [1]. One branch of target-based marketing focuses on recommender systems, which track past purchases of users, try to learn user preferences over time, and automatically recommend products that fit customer needs. Recommender systems proved to be very important tools for big companies such as Amazon, Netflix, and MovieLens.

Recommender systems are also used in the context of online social networks such as Facebook, Orkut and MySpace. Such online sites try to recommend 'good' friends to users to keep them engaged and thus have them spend more time on the site.

Most work in this area so far, however, has considered only the purchase/link probability and user preferences to make recommendations. This is convenient from the users' point of view as recommendations with high purchase/link probabilities are likely to satisfy the users. On the other hand, marketing strategies should satisfy both the needs of users as well as provide a higher 'gain' to the vendors. Here, increasing the gain of the vendors should obviously not mean to recommend the most expensive (='valuable') products to customers in an online shopping site, for example. Value-oriented recommendations should be made carefully such that the user is kept satisfied. Otherwise the users could lose trust in the recommendations and quit.

In this paper, we argue that the recommender systems should be designed in a way such that both the value of recommendations to the vendors (=vendor gain) and the purchase/link probability of users (=user satisfaction) can be integrated into the recommendation framework to properly balance the views between the users and the vendors.

In traditional recommender systems, recommendations to a certain node (e.g. a user) i are made based on the network 'proximity' of other nodes (e.g. another user or a product) to i. Often top k nodes with the highest proximity to i are recommended (network proximity is a.k.a. closeness, relevance, similarity). In this type of recommendation schemes, only the network structure is exploited to compute the so-called proximity scores. Often, additional knowledge about the users such as their age, interests, life routines in social networks or similarly the price of products in customer-product networks can not be directly incorporated into the system. Next we give several example scenarios to show why we think incorporating such knowledge is an important task.

For example, consider the case in a social network setting where the system can determine the 'network-value' of its users, which is a measure of centrality (central users are those who have small average distance to the rest of the users in the network, i.e. they can reach everyone in a small number of steps on average). Then, the system can recommend 'friends' to a user i who are *both* in high proximity to i, *as well as* have high centrality. This is a win-win situation for both the users and the system: users in a social network have high incentive to link to close-by *and* central users as a user i gets 'closer' to others by becoming friends with central users. More intuitively, say in

a professional network, people in fact want to meet central people since that helps them meet others more easily and build their own professional network faster. In addition, users linking to central users make the network tighter and better connected, with a small diameter [2]. In such networks, everybody can reach one another in a small number of steps and the system is more resilient to users leaving the network. Here, the 'value' of a recommendation is referred to as 'centrality' and the system 'gain' as 'network resilience'.

Another example of integrating 'value' into the recommendation process is when the system has meta-data of users such as their personal interests, political views, etc. Based on this knowledge, it could recommend 'friends' to a user i, who are *both* in high proximity *as well as* have high personal similarity to i. This again is a win-win situation for both the users and the system: users in a social network have high incentive to link to close-by *and* similar users as they are the 'familiar strangers' with common interests. In addition, ties between similar users as are expected to be strong ties and thus yield a more robust network. Here, the 'value' of a recommendation is referred to as 'user similarity' and the system 'gain' as 'network robustness'.

Finally, a more intuitive example can be given based on customer-product networks. Consider the case in such a network where the system has and wants to incorporate the price of products into the recommendation process. Then, it can recommend products to a customer i that are *both* in high proximity to i, *as well as* have high price. Here the proximity ensures that the customer will be presented reasonably 'good' products that s/he will be satisfied with and close-by *and* high price products ensures higher vendor profit margin (hence a win-win situation). Here, the 'value' of a recommendation is referred to as 'price' and the system 'gain' as 'profit'.

In this paper, we propose `ValuePick`, a recommendation framework that meets both goals as discussed in the examples above. The main contributions of our work can be listed as follows:

- **Problem formulation:** We formulate the problem of generating recommendations as an optimization problem, in order to properly balance the viewpoints of both parties: the users and the system. Our formulation enables incorporating the systems' external knowledge (i.e., value of recommendations) into the recommendation process such that higher gain is achieved while the user satisfaction is kept high (Section 3).
- **Design of our method:** Our method is *parsimonious* (it only requires *one* parameter, $\tau \in [0, 1]$ for tolerance), *flexible* (the parameter τ gives users the flexibility to experiment: if the user tolerance τ is 0, `ValuePick` defaults to the conservative policy and make recommendations *by proximity only* (without incorporating any 'value'). Whereas, if the user is willing to experiment and user tolerance τ is 1, then `ValuePick` makes the most 'valuable' recommendations that would yield the highest gain to the system), and *general* (it can be used with any 'value' metric, e.g. centrality, similarity, price, and so on).
- **Performance study:** We perform experiments on the 'dual-goal link recommendation in social networks' problem, run comprehensive simulations on two real networks, compare our results with other recommendation strategies and basic heuristics, and show that our method yields well-connected networks with reasonably high user satisfaction, the level of which can also be adjusted (by changing τ) (Section 4).

We conclude our paper with a survey on related work in Section 5 and conclusions in Section 6. See Table I for the list of notation used throughout text.

Symbol	Description
$\mathcal{G}$	Graph representation of a dataset
N	Number of nodes in $\mathcal{G}$
E	Number of edges in $\mathcal{G}$
$p_{i,j}$	proximity between nodes i and j in $\mathcal{G}$
$\mathbf{r}$	the 'value' vector of nodes
$\mathbf{x}$	the binary solution vector, $\mathbf{x}(\mathbf{i}) \in \{0, 1\}$ in `ValuePick`.
k	Number of nodes recommended to each node in $\mathcal{G}$.
τ	parameter in `ValuePick` to control perturbation. $\tau \in [0, 1]$

Table I
TABLE OF SYMBOLS USED IN THE PAPER.

II. BACKGROUND AND PRELIMINARIES

In our experiments, we used two "proximity" metrics: (1) the Random-Walk-with-Restart (RWR)-based Pagerank scores, and (2) the Katz scores. Also, as the "profit" of a node in a social network, we used (1) the betweenness centrality, and (2) the eigenvector centrality.

We emphasize that in our experiments, instead of using the direct centrality scores, we define a new score called 'recursive' centrality $\mathbf{r}$. To compute $\mathbf{r}$, we multiply the all-pairs-proximity matrix $\mathbf{R}$ with the direct centrality vector $\mathbf{r}_{dir}$, i.e. $\mathbf{r} = \mathbf{R} * \mathbf{r}_{dir}$. Intuitively, the 'recursive' centrality score is high for those nodes which have both high direct centrality *as well as* high proximity to other central nodes.

III. OUR METHOD `ValuePick`

In this section we first give the intuition behind our proposed method by a running toy example, and then we formulate the *dual-goal* recommendation problem as an optimization problem which can be solved by using integer programming.

A. The toy example

1) Problem specification:: Consider the toy graph in Figure 1(a) in which assume that the vertices represent users and edges represent friendship links. Top 10 users with

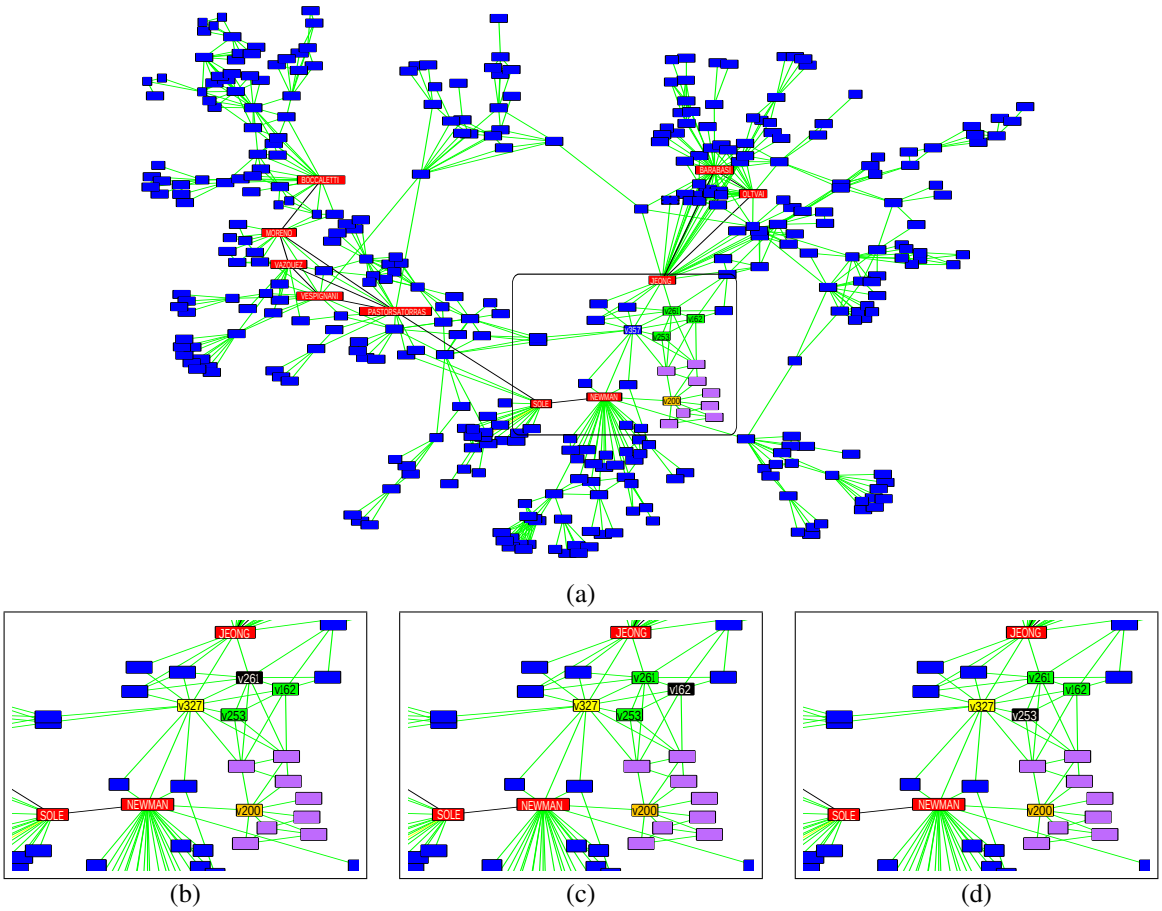


Figure 1. (top) The toy social network of users and friendship links. Annotated red nodes show the top 10 users with the highest 'betweenness centrality'. Top 3 recommendations (green) to vertex $v200$ (orange) w.r.t. (a) proximity only (tolerance τ=0, baseline), (b) 8% more, (c) 12% more, and (d) 16% more expected gain than the baseline, with increasing τ. Purple nodes show the direct neighbors of $v200$. Figure best viewed in color.

the highest betweenness centrality are colored in red and annotated.

Here, we want to make recommendations to the users, that is, we want to suggest new friends, say to user i, the goal being twofold: First, the recommended users should be among those with high proximity to i. Second, the recommended users should be among those with high 'centrality' scores such that the new links should make user i get 'closer' to the rest of the nodes in the network. In short, the recommendation system should return the top k recommendations to user i that (1) are likely to occur and so will satisfy the user i, and (2) have high 'recursive centrality' (Section II).

2) Recommendations by proximity only:: Let us consider the vertex $v200$ in Figure 1(a) shown in orange color. If we start from $v200$, and do a random walk with some restart probability (RWR), the steady state probabilities of landing at each vertex (Pagerank score) gives the proximity scores of all the vertices w.r.t. the starting vertex $v200$. If we are to make 3 recommendations, top 3 vertices with the highest Pagerank scores would be $v253$, $v162$ and $v261$, The *expected gain* of linking to these vertices, which is (proximity*centrality) (assuming linking probability is proportional to proximity), is shown in green in Figure 1(a) with corresponding proximity scores shown in Table II. also shown in the last column. We will consider this set of vertices as our baseline recommendations.

vertex ID	RWR-proximity	Expected Gain(prox*RBC)
253	0.0255	6.76687e-005
162	0.0253	7.788615e-005
261	0.0241	8.703078e-005
327	0.0233	0.00010628
165	0.0102	2.167184e-005

Table II
TOP 5 VERTICES WITH THE HIGHEST RWR PROXIMITY SCORES TO VERTEX $v200$ IN THE COLLABORATION NETWORK OF SCIENTISTS IN FIGURE 1(A) AS WELL AS THEIR RWR-PROXIMITY SCORES. THE LAST COLUMN SHOWS THE EXPECTED GAIN TO $v200$ FROM GETTING LINKED TO THESE VERTICES, WHICH IS (RWR-PROXIMITY*RECURSIVE BETWEENNESS CENTRALITY(RBC)).

3) Integrating 'value':: As we can see from Table II, $v327$ which has the 4^{th} rank in proximity to $v200$ has the

highest expected gain. This is intuitive, $v327$ is both in close neighborhood of $v200$ and is connected to two highly central nodes, *Newman* and *Jeong* in the graph. The main idea behind our proposed method is that if we are allowed to perturb the rank-list of vertices by proximity, that is the first column in Table II, we can recommend another set of 3 vertices with greater total expected gain. Of course, this perturbation should not be drastic such that the trust of the customer is compromised, i.e. even though the expected gain of a vertex that is ranked say 1000^{th} by proximity is the highest, we should not be recommending it in top 3 recommendations as it might not well satisfy the user. On the other hand, in our toy example the vertex with the highest expected gain is ranked 4^{th} by proximity, so if we switch the places of vertices with ranks 3 and 4, and recommend $v253$, $v162$ and $v327$ instead, we end up with greater expected gain (in particular 8% more gain over the baseline) with little perturbation. See Figure 1(b).

The amount of gain by perturbation in rankings depends on how much we are willing to compromise the original ordering of vertices. If we are willing to compromise more of user satisfaction/trust, then instead of switching $v261$ with $v327$, we can switch $v162$ with a lower expected gain than $v261$ (Figure 1(c)). Here, we obtain 12% more gain over the baseline, but drop the 2^{nd} ranked vertex instead of the 3^{rd} in place of $v327$. Finally, if we increase the perturbation tolerance even more, we can flip $v327$ for $v253$ which is ranked 1^{st} and obtain 16% more gain over the baseline which is the best we can do (Figure 1(d)). This is also intuitive because even though $v253$ is in the very close neighborhood of $v200$, it is farther from the central vertices in the graph compared to $v261$, $v327$ and $v162$ that are only one or two hops away to the most central vertices.

B. Definition and Formulation of the Meta-problem

In this paper we focus on the dual-goal recommendation problem in the context of social networks. However, the ideas can be generalized to any type of network with any given 'value' metric. The meta-problem we solve is, *given a graph $\mathcal{G}$ and a node i in this graph, which k nodes should we recommend to i such that node i is reasonably satisfied with the recommendations as well as the network becomes tighter with shorter average distance between nodes.* The heart of our solution can be stated as "perturbing the proximity-ranklist of vertices in order to increase the total expected gain", which formulate as an optimization problem.

Here we assume that the 'value' of linking to a vertex i is denoted as its 'recursive centrality' r_i. (Again the 'value' can be any other metric; in the experiments we used both 'betweenness' and 'eigenvector' centrality scores). We also assume that the probability of vertex i to accept to link to a recommended vertex j is the proximity $p_{i,j}$ between i and j. (Again the proximity can be any preferred measure; in the experiments we used both Katz and RWR proximities).

Then, our objective function is to maximize the total expected gain from the k recommended vertices, which is the sum of the multiplication of value r and proximity p of those k vertices. Table III(1a) denotes the objective function of the problem, where $\mathbf{x}$ is a vector of size N, the number of vertices. In this vector, $\mathbf{x}(\mathbf{i})$ can take values of either 1 or 0 (Table III(1b)), being 1 if vertex i is in the solution set and 0 otherwise. Also since we want to make k recommendations, the values in vector $\mathbf{x}$ should sum to k (Table III(1c)).

$$\underset{X}{\text{maximize}} \quad \sum_{i=1}^{N}(\mathbf{r}(\mathbf{i}) * \mathbf{p}(\mathbf{i})) * \mathbf{x}(\mathbf{i}) \tag{1a}$$

$$\text{subject to:} \quad \mathbf{x}(\mathbf{i}) \in \{0, 1\}, \tag{1b}$$

$$\sum_{i=1}^{N}\mathbf{x}(\mathbf{i}) = k, \tag{1c}$$

$$\frac{\sum_{i=1}^{N}(M - \mathbf{p}(\mathbf{i})) * \mathbf{x}(\mathbf{i})}{k} \leq \tau. \tag{1d}$$

Table III
THE MAIN PROBLEM FORMULATION OF `ValuePick` TO MAKE k RECOMMENDATIONS TO A GIVEN VERTEX. $\mathbf{r}$ IS THE GLOBAL PROFIT VECTOR AND $\mathbf{p}$ IS THE PROXIMITY VECTOR OF ALL THE VERTICES WITH RESPECT TO THE GIVEN VERTEX. $M = \frac{\sum_{i=1}^{k}\hat{\mathbf{p}}(\mathbf{i})}{k}$, IS THE MEAN OF THE TOP-k VALUES IN THE SORTED LIST OF PROXIMITIES $\hat{\mathbf{p}}$. $\mathbf{x}(\mathbf{i})$ IS 1 IF VERTEX i IS IN THE SOLUTION SET, AND 0 OTHERWISE. τ IS THE TOLERANCE WHICH CONTROLS THE AMOUNT OF DEVIATION FROM THE ORIGINAL RANKING OF VERTICES. FINALLY, N IS THE TOTAL NUMBER OF VERTICES IN THE GRAPH.

As is, that is without any further constraints, the solution is the set of the top k vertices in the sorted list of vertices *by expected gain*, $\mathbf{r}*\mathbf{p}$. However, this set might be too different than the set of top k vertices in the sorted list of vertices *by proximity only*, and as a result might not satisfy the user and cause him/her lose trust. Therefore, we introduce another constraint to control the amount of perturbation w.r.t. the original ranklist $\mathbf{p}$ so as to retain trust at high levels. In Table III(1d), $\frac{\sum_{i=1}^{N}(M-\mathbf{p}(\mathbf{i}))*\mathbf{x}(\mathbf{i})}{k} \leq \tau$ states that the total proximity scores of the returned set of k vertices should not deviate too much from the total proximity scores of the top k vertices in the original ranklist $\mathbf{p}$ and is adjusted by parameter $\tau \in [0, 1]$, which we call as the 'perturbation tolerance'. If τ is set to 0, that is if tolerance is zero, the top k vertices with the highest proximity scores are returned as they make $\sum_{i=1}^{n}(M - p(i)) * \mathbf{x}(\mathbf{i})$ equal to 0. On the other hand if τ is 1, then one could resort to the k vertices that would yield the highest *expected gain*. So, we note that the two extreme values of τ give the two special cases of recommendations; (1) the traditional, customer-oriented method which considers recommending *by proximity only*, and (2) the vendor-oriented method which considers recommending *by maximum expected gain*. τ in `ValuePick` properly balances the both extremes.

1) Solution to the optimization problem:: As formulated, the link recommendation task above is similar to the famous 0-1 Knapsack problem [3]. Informal description of the 0-1 Knapsack problem is that 'given a set of n items, each with a weight w_i and a value v_i, find a subset of items so that the total weight of the items in the subset is less than a given limit W and their total value is as large as possible.' On the same lines, we want to find a subset to recommend from N vertices, each with weight w_i that corresponds to $(M - p_i)$, a value v_i as $(p_i * r_i)$ and a total weight limit W as τ, such that the total value of the subset is maximized. The problem as described is NP-complete, however, there exists a pseudo-polynomial algorithm that uses dynamic programming that runs in $O(nW)$ time and uses $O(nW)$ space. We note that in our setting we have an additional constraint: we want the size of the subset to be k (Table III(1c)). As a result it becomes a harder optimization problem. In our experiments, we use the CPLEX [4] optimization package to solve linear and integer programming problems. In our experiments, we notice that the running time of CPLEX in practice is reasonably small. We discuss the computational cost of `ValuePick` in Section 4.5 in more detail.

IV. Empirical Study

In marketing, its often hard to predict the effect of an intervention in the marketing scheme. The intervention in our case is the perturbation of the rankings in the original list of recommendations sorted by proximity. Here, it is not very clear how users will respond to these 'adjustments'. That is, it is hard to predict how user satisfaction will be affected in advance before the method is deployed. One advantage of our method, on the other hand, is that it provides the tolerance τ parameter to flexibly set the 'level-of-adjustment' for each user. Empirical study [5] shows that users can rate the same item differently at different times, which indicates that users already have natural variability in their decisions. This supports our conjecture that controlled perturbation in the order of recommendations should not affect the user decisions drastically.

As it is not possible for us to deploy our method in a real setting, we resort to simulations in this section. In our experiments we used two real networks:

- *Netscience*: the collaboration network of network theory scientists compiled from the bibliographies of two review articles by M.Newman and S. Boccaletti et. al., respectively. There exists a link between two scientists if they have co-authored a paper. The giant connected component (GCC) of this network contains 379 nodes, and 914 edges.
- *DBLP*: the co-authorship network of authors in DBLP, between the years 1959 and 1979. There exists 931 nodes and 1705 edges in the GCC.

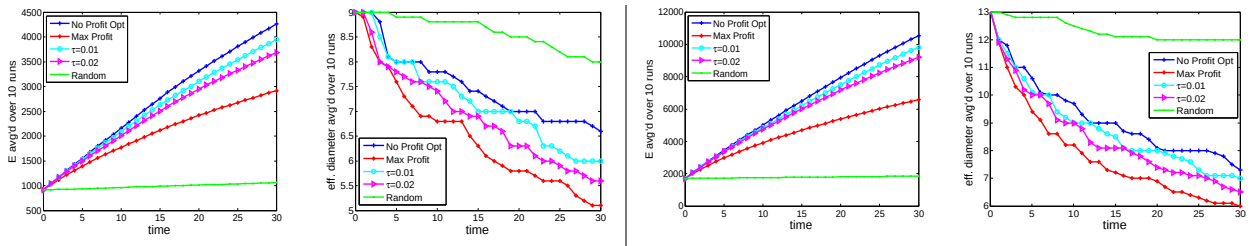
In our simulations, we used *four* different recommendation strategies: (1) *NoGainOpt*: recommend the top k nodes with the highest proximity score; (2) *MaxGain*: recommend the top k nodes with the highest expected gain, i.e. (proximity*value) score; (3) `ValuePick`: recommend the best k nodes which maximize the total (proximity*value) score such that average perturbation in proximity scores is less than τ; and (4) *Random*: recommend k nodes randomly. (Note that *NoGainOpt* and *MaxGain* are the special cases of `ValuePick` with τ=0 and τ=1, respectively). As we stated in Section 2, we experimented with two proximity metrics (Katz and RWR scores) and two network-value metrics (betweenness and eigenvector centrality scores). Here, we assume that the probability that node i will accept to link to a recommended node j is proportional to the proximity between i and j.

Specifically, our experimental set-up is as follows: (Step 1) At each time step T, we make k recommendations to each node i in the graph using one of the four methods described above. (Step 2) Node i links to a recommendation j with probability proportional to the proximity between i and j. (Step 3) After all nodes are processed, since the graph structure changes, the proximity scores as well as the 'recursive centrality' scores are recomputed, and the simulation goes back to Step 1. In our experiments we used different values for T and k, but we report our results for T=30 and k=5 for brevity.

For evaluation, we track the total number of recommendations accepted (as an indicator of user satisfaction), as well as the effective diameter of the network over time (as an indicator of 'gain'). Intuitively, the effective diameter represents how much of a "small world" the network is – how quickly one can get from one 'end' of the network to another.

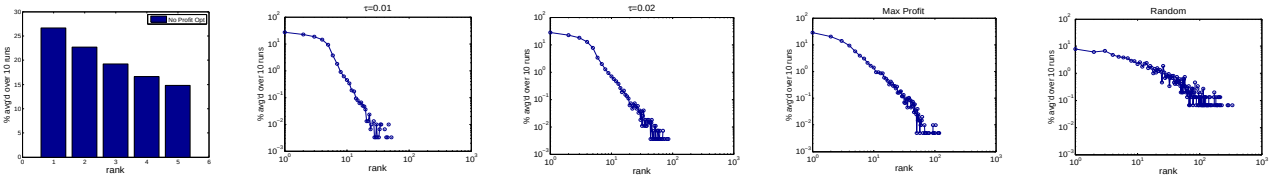
A. Results using RWR-proximity and Betweenness Centrality

In Figure 2 we show our first set of results for (left) *Netscience* and (right) *DBLP* (Note that all the results are averaged over 10 runs). In particular, (a) shows the number of edges E in the network over time. We observe here that both graphs reach the highest number of edges when *NoGainOpt* is used. This is intuitive, because in this scheme the centrality of nodes being recommended is not considered and the 'closest' nodes are recommended. As we take the link probability proportional to proximity, *NoGainOpt* yields the most number of links accepted. We also observe that E drops considerably when *MaxGain* is used, since the perturbation when only the expected profit is considered might be high and thus could cause users to be unsatisfied. On the other hand with `ValuePick`, E stays somewhere in the middle, compromising user satisfaction only partially. Also notice here that, we ran simulations using different values for tolerance τ. We can observe that user satisfaction can be increased by choosing a smaller τ. Finally, notice that *Random* is the least acceptable method as expected.



(a) E vs. time (b) eff.diam. vs. time (a) E vs. time (b) eff.diam. vs. time

Figure 2. Results on (left) *Netscience* and (right) *DBLP*, using RWR-proximity and betweenness-centrality-profit, T=30, k=5, averaged over 10 runs. (a) number of edges E and (b) effective diameter versus time are shown. Blue pluses represent *NoGainOpt*, red stars *MaxGain*, cyan circles `ValuePick`(τ=0.01), pink triangles `ValuePick`(τ=0.02), and green dots *Random*. Notice that our proposed methods (τ=0.01 etc) achieve a balance: high user satisfaction (#edges), and at the same time, high vendor profit (small diameter). Figure best viewed in color.



(a) *NoGainOpt* (b) τ=0.01 (c) τ=0.02 (d) *MaxGain* (e) *Random*

Figure 3. Ranks w.r.t. proximity of nodes that are linked to via recommendations for different methods (from left to right) *NoGainOpt*(τ=0), `ValuePick`(τ=0.01), `ValuePick`(τ=0.02), *MaxGain*(τ=1)and *Random*, over 10 runs in *Netscience*. Results for *DBLP* are similar and thus omitted. Notice that the ranks get wider in range from left to right with increasing perturbation tolerance of the methods.

Figure 2(b), on the other hand, shows the effective diameter over time. We observe that the diameter shrinks earlier for profit-oriented methods *MaxGain* and `ValuePick` even though at any given time the graph density is the highest for *NoGainOpt*.

Observation A: `ValuePick` *successfully achieves a balance between user satisfaction, with high number of recommendations accepted, and vendor gain, with a small diameter which shrinks early in time.*

The results using Katz-proximity and Eigenvector-centrality are similar and so we exclude them for brevity.

B. Analysis of ranks

Next, we examine the ranks (in the sorted list of nodes *by proximity*) of nodes to which new links were formed (recommendations that were accepted) for all four recommendation strategies, averaged among all the nodes over 10 runs.

Figure 3 shows the % fraction versus the rank of such links for (from left to right) *NoGainOpt*, `ValuePick` with $\tau = 0.01$ and $\tau = 0.02$, *MaxGain*, and *Random*. As the perturbation tolerance of the methods *increase* from left to right, the range of ranks also gets wider from left to right. Notice that in *Netscience*, by using `ValuePick` with low tolerance $\tau = 0.01$, links to nodes with rank up to 60 is formed, while with *MaxGain* with tolerance $\tau = 1$, the highest rank goes over 100.

Here we note that in the simulations we used the same τ for all the users in the network while making recommendations. In reality, however, τ can be chosen differently for each user depending on how much a user is tolerant to experiment. It can even be changed dynamically depending on the feedback from each user. For example, if the user starts rejecting the recommendations for a certain τ, then τ can be lowered for that user; whereas if the user seems to be satisfied, τ can be increased more for higher vendor gain.

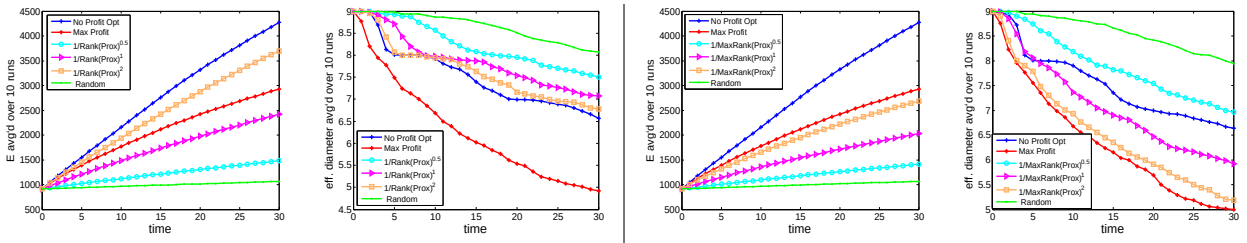
Observation B: *The tolerance parameter τ in* `ValuePick` *successfully controls the amount of perturbation, and thus the rank-range of recommended nodes in the original rank-list by proximities.*

C. Comparison to other heuristics

Given the idea behind `ValuePick`, is there a simpler/faster way to introduce perturbations? Here we show that this may be difficult: two simple heuristics, *Rank* and *MaxRank*, we describe below do *not* perform well.

Particularly in *Rank*, to recommend k nodes to a given node i, we start from the top of the list of nodes sorted *by their proximity* to i, flip a coin and recommend that node with probability $\frac{C}{Rank^{\alpha}}$, where C is the normalization constant so that probabilities sum to 1. For example, the node in the top of the list is recommended with probability C, the second one with probability $\frac{C}{2^{\alpha}}$, and so on. *MaxRank* is similar but the list of nodes is sorted *by their (proximity*value)*. As one can notice, both of these methods also introduce randomized perturbations to the original rank list of recommendations as in `ValuePick`.

In Figure 4 we show the results of (left) *Rank* and (right) *MaxRank* for *Netscience* (results look similar for *DBLP*and thus omitted here for brevity). Here, we used $\alpha = \{0.5, 1, 2\}$; the higher the α, the narrower the range



(a) E vs. time (b) eff.diam. vs. time (a) E vs. time (b) eff.diam. vs. time

Figure 4. Results on *Netscience* with (left) $\frac{C}{Rank^{\alpha}}$, and (right) $\frac{C'}{MaxRank^{\alpha}}$ for α={0.5,1,2} T=30, k=5, over 10 runs. (C and C' are the normalization constants so that probabilities sum to 1.) Results for *DBLP* are similar and thus omitted.

of ranks of the recommended nodes gets. For example in Figure 4(left) *Rank*(α=2) is closer to *NoGainOpt* and (right) *MaxRank*(α=2) is closer to *MaxGain*. We notice that even though all *Rank*(0.5,1,2) yield less E (user acceptance) than *NoGainOpt*, they cannot shrink the effective diameter as much as *NoGainOpt*. That is, there is no return in place to less user satisfaction. Similarly, all *MaxRank*(0.5,1,2) yield less E than *MaxGain*, whereas *MaxGain* gives the shortest effective diameter.

Observation C: *Two simple heuristics, though introduce perturbations in a simpler way, do not balance user-satisfaction and gain properly.* `ValuePick`*, on the other hand, fairly trades user acceptance as an exchange to better network connectivity.*

D. Analysis of computation time

In this section we provide a run-time analysis of the CPLEX optimization tool that we used to solve the integer programming problems in `ValuePick`. In Figure 5(a) we show the time it takes to find the best k=5 nodes to recommend to each node out of 379 nodes for (left) *Netscience* and 931 nodes for (right) *DBLP*. We notice that it only takes ~ 0.047 seconds to solve an integer programming problem with around $1K$ variables with CPLEX for *DBLP*. This value is the average over T=30 time-steps times 10 runs, i.e. 300 trials per node.

In our simulations, we recommended k=5 nodes to *all* the nodes in the graph. Then, for example for *DBLP* the expected run-time for 1 run in which k recommendations are made to all 931 nodes over T=30 iterations is $0.047 * 931 * 30 \approx 22$ minutes. We show the total CPLEX run-time over 30 iterations for both (left) *Netscience* and (right) *DBLP* in Figure 5(b). The run-time is 22.3 minutes for *DBLP* and 14 minutes for *Netscience* with 379 nodes. (All experiments were performed on the same machine with a 3.2GHz Pentium CPU and 2GB memory.)

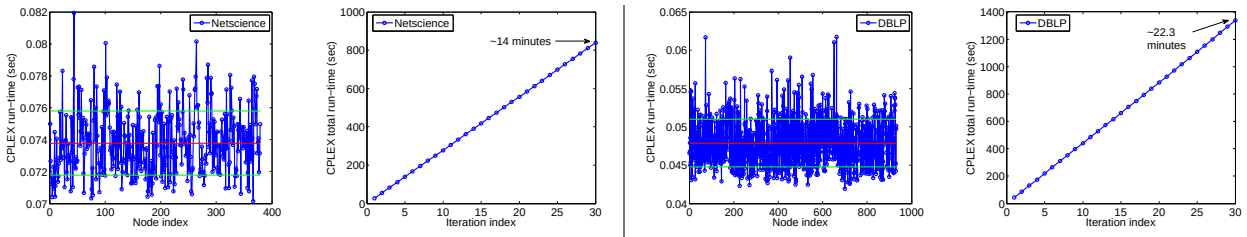
Observation D: *A fast solution to* `ValuePick` *exists with current optimization tools such as CPLEX. Finding the solution set with up to $1K$ variables takes about $\frac{1}{20}$ of a second in practice.*

V. Related Work

There has been a wide range of research on recommender systems and how to improve their recommendation quality, however, much less has been done towards integrating 'value' into the system.

Brand [6] uses random walk based measures to compute proximity in a customer-movie graph in which the goal is to maximize the 'expected discounted profit' by "nudging" customers into states from which the random walk will pass through more profitable states earlier in the walk. [7] also considers the profitability of products in recommender systems. Their method is, however, solely based on maximizing 'expected profit' which does not explicitly take into account user satisfaction. More recently, Das et. al. [8] proposed a method to maximize vendor profit while providing trustworthy recommendations to the customers. They did not however conduct the empirical study. Anagnostopoulos et. al. [9] also study the problem of query recommendation in an optimization framework. In their setting there is no explicit definition of vendor profit but only user satisfaction.

In their award-winning paper on the Netflix competition [10] use factor models, among other concepts, but they also do *not* consider profit in their formulation. Latent factor models typically involve solving a non-convex optimization (possibly using an EM-style algorithm or Alternating Least Squares). These approaches usually require good initialization (or several random restarts) and has no guarantees on the quality of the solution obtained. On the other hand, we propose an Integer Programming (IP) approach to solve the problem using CPLEX. CPLEX uses branch-and-cut algorithms to solve these IP problems and in most cases returns the optimal solution (along with a certificate of optimality). In the cases when CPLEX does not find an optimal solution it returns the value of the best solution found so far along with a bound on the distance of this solution from the optimal solution (either using an LP relaxation or a dual certificate). We would also hasten to point out that for the problems we discuss in the paper and for the graphs we are interested in practically CPLEX has always returned the optimal solution (indicating that typical graphs do not result in hard IP problems).



(a) time vs. nodes (b) total time vs. iter.s (a) time vs. nodes (b) total time vs. iter.s

Figure 5. (a) Run time of CPLEX (in seconds) for each node averaged over 30 time-iterations x 10 runs = 300 trials (left) in *Netscience*, and (right) in *DBLP*. Red lines show the average (~ 0.074 and ~ 0.047 sec.s,respectively), green lines show the 1 standard deviation below and above. (b) Total run time of CPLEX for *all* the nodes for 30 iterations averaged over 10 runs. Note that 1 run of `ValuePick` takes about 14 minutes for (left) *Netscience*, and 22 minutes for (right) *DBLP* with 379 and 931 nodes, respectively.

VI. Conclusions

This paper is one of the few work on recommender systems that focuses on integrating 'value' of recommendations into the system in order to increase expected gain for the vendors while keeping user satisfaction at high levels. Our contributions can be summarized as follows:

1) **Problem formulation:** We formulated the problem of link recommendation as an optimization problem and proposed `ValuePick`, which incorporates the 'value' of recommendations into the system.
2) **Design of our method:** `ValuePick` is (a) *parsimonious* –requires only *one* parameter, tolerance τ; (b) *flexible* –τ can flexibly be set to properly balance the trade-off between maximizing expected gain and achieving high user satisfaction. Two straightforward schemes, recommendation by proximity only (τ=0) and recommendation by maximum expected gain (τ=1), are the two special cases of `ValuePick`; and (c) *general* –it can be used with any 'value' score.
3) **Performance study:** We ran extensive simulations using two real collaboration networks, compared our method to two other schemes *Rank* and *MaxRank* that use simple heuristics, and showed that `ValuePick` gives the best results satisfying the *dual-goal* of obtaining a well-connected network with small diameter and keeping user satisfaction at high levels. Experiments show that CPLEX provides a fast solution to `ValuePick` in practice.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. IIS0808661, iCAST and under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the funding agencies, the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work is also partially supported by an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

References

[1] D. M. Fleder and K. Hosanagar, "Recommender systems and their impact on sales diversity," in *ACM conference on Electronic Commerce*, 2007, pp. 192–199.

[2] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD*, 2005, pp. 177–187.

[3] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[4] IBM-ILOG, "Cplex optimization package," http://www-01.ibm.com/software/integration/optimization/cplex/.

[5] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *CHI*, 1995, pp. 194–201.

[6] M. Brand, "A random walks perspective on maximizing satisfaction and profit," in *SIAM*, 2005.

[7] L.-S. Chen, F.-H. Hsu, M.-C. Chen, and Y.-C. Hsu, "Developing recommender systems with the consideration of product profitability for sellers," *Inf. Sci.*, vol. 178, no. 4, pp. 1032–1048, 2008.

[8] A. Das, C. Mathieu, and D. Ricketts, "Maximizing profit using recommender systems," in *WWW*, 2010.

[9] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis, "An optimization framework for query recommendation," in *WSDM*, 2010, pp. 161–170.

[10] Y. Koren, "Collaborative filtering with temporal dynamics," in *KDD*, 2009, pp. 447–456.