Ranking Information in Networks

Tina Eliassi-Rad[†‡] Keith Henderson[‡]
[†]Rutgers University [‡]Lawrence Livermore Lab
tina@eliassi.org keith@llnl.gov

August 5, 2010

Introduction

Given an information network, we are interested in ranking sets of nodes that score highest on user-specified criteria. Examples include (1) discovering sets of authors with expertise in a wide range of disciplines; (2) finding sets of patents which, if removed, would have the greatest effect on the patent citation network; (3) identifying small sets of IP addresses which taken together account for a significant portion of a typical day's traffic; and (4) detecting sets of countries whose vote agree with a given country (e.g., USA) on a wide range of UN resolutions.

We present this ranking task as a *Top-K* problem. We assume the criteria is defined by L real-valued features on nodes. This gives us a matrix F, whose rows represent the N nodes and whose columns represent the L real-valued features. Examples of these node-centric features include degree, betweenness, PageRank, etc. We define an *L-tuple* $\langle v_1, \ldots, v_L \rangle$ as a selection of one value from each column. The score of an L-tuple is equal to the sum of the selected values (i.e., sum of the tuple's components). For a given parameter K, we require an algorithm that efficiently lists the top K L-tuples ordered from best (highest score) to worst. The L-tuple with the highest score corresponds to a set of nodes in which all features take on optimal values.

As described in the next section, we solve the Top-K problem by utilizing SMA* [1], which is a fixed-memory heuristic search algorithm. SMA* requires the specification of an additional parameter M for the maximum allotted memory-size. The choice for M has a dramatic effect on runtime. To solve this inefficiency problem, we introduce a parallelization of SMA* (on distributed-memory machines) that increases the effective memory size and produces super-linear speedups. This allows use to efficiently solve the Top-K ranking problem for large L and K (e.g., $L \in [10^2, 10^3]$ and $K \in [10^6, 10^9]$). Experiments on synthetic and real data illustrate the effectiveness of our solution.

A Serial Solution to the Top-K Ranking Problem

Given F, a matrix of L columns of real-valued features over N rows of vertices, and an integer K, our application of SMA* will report the top K scoring L-tuples that can be selected from F. First, we take each column of matrix F and represent it as its own vector. This produces a list X which contains L vectors (of size $N \times 1$). We sort each vector in decreasing order. Then, we begin constructing D, another list of L vectors of real numbers. Each vector D_i has $|X_i| - 1$ entries, where $D_{ij} = X_{ij} - X_{i(j+1)}$. Note that the values in each D_i are all nonnegative real numbers; and D_i is unsorted.

The top-scoring L-tuple, R_1, can immediately be reported; it is simply the tuple generated by selecting the first element in each X_i. At this point, the problem can be divided into two subproblems whose structure is identical to the original problem (i.e. another instance of the Top-K problem). While there are several possible ways to make this division, we choose one that allows us to search efficiently. In particular, we choose the vector which would incur the least cost when its best element is discarded. This can be deter-

mined by choosing the vector i with the minimum D_{i1}. Given this index i, we generate two subproblems as follows. In the first subproblem, we discard the best element in list X_i. The resulting list of vectors will be called X^1. In the second subproblem, we keep the best element in vector X_i and remove all other elements from vector X_i. The resulting list of vectors here will be called X^0.

Let's illustrate this procedure with an example. Suppose X = {[10,8,5,2,1], [4,3,2,1], [30,25,24,23,22]}; so, D = {[2,3,3,1], [1,1,1], [5,1,1,1]}. (Recall that we assume lists in X are already sorted in decreasing order and some entries maybe missing.) The top-scoring L-tuple is $R_1 = \langle 10, 4, 30 \rangle$ with $score(R_1)$=10+4+30=44. Starting from X, the next best tuple can be generated by selecting the list i with the smallest D_{i1} and decrementing X_i in X: X^1 = {[10,8,5,2,1], [3,2,1], [30,25,24,23,22]}, D^1 = {[2,3,3,1], [1,1], [5,1,1,1]}, and $score(X^1)$=10+3+30=43. At this point, we can split the problem into two smaller problems. We can either "accept" the best decrement (X^1 above) or "reject" the best decrement and all future chances to decrement that list: X^0 = {[10,8,5,2,1], [4], [30,25,24,23,22]}, D^0 = {[2,3,3,1], [], [5,1,1,1]}, and $score(X^0)$=10+4+30=44.

Our procedure for generating subproblems has three important properties. First, every L-tuple generated from X is either R_1, from X^1, or from X^0. Second, no L-tuple generated from X^1 or X^0 has a score greater than $score(R_1)$. Third, the top-scoring L-tuple from X^0 has the same score as R_1. Given this formulation, a recursive solution to the Top-K problem is possible. In the base case, the input list X has L vectors of length one, and there is only one possible L-tuple to report (namely, R_1). Given arbitrary length vectors in X, we use SMA* [1] (a heuristic search approach), which allows us to generate a the top K tuples one at a time and satisfy our space complexity of less than $O(K)$.

A Parallel Solution to the Top-K Ranking Problem

The aforementioned serial algorithm is very sensitive to the choice of M, the maximum amount of memory that can be allocated. Here we present a parallel SMA* algorithm that offers dramatic improvement in runtime.

For a machine with P processing nodes, we use A* search to generate the P–1 best candidate subproblems. This covers the entire search tree. Because each subproblem is independent of the others, each processing node can perform SMA* search on its subproblem and incrementally report results to the master processing node, where the incoming tuple lists are merged and results are reported.

For small P, this algorithm works as expected and produces super-linear speedup. However as P increases, the performance boost can decrease quickly. This occurs because the runtime for this parallel algorithm is dominated by the single processing node with the longest runtime. If the initial allocation of subproblems to processing nodes is imbalanced, additional processing nodes may not improve performance at all. To ameliorate this problem, we adopt a load-balancing heuristic.

We run parallel SMA* on the input data with $K' \ll K$ as the threshold parameter. We then use the relative load from each subproblem as an estimate of the total work that will have to be done to solve that subproblem. We use these estimates to redistribute the initial search-tree nodes and repeat until there are no changes in the initial allocation of search-tree nodes. This distribution is then used to generate the top K L-tuples as described above. In our experiments, this heuristic correctly balanced the loads on the processing nodes. Moreover, the initial overhead to calculate the estimates was a negligible fraction of the overall runtime.

Experiments

We tested our Top-K algorithm on synthetic and real-world (namely, patents, IP traffic, DBLP, and UN voting) data sets. For brevity, we only discuss one of them here. Our patents data set is a collection of 23,990 patents from the U.S. Patent Database (subcategory 33: biomedical drugs and medicine). The goal here is to discover sets of patents which, if removed, would have the greatest effect on the patent citation network. In particular, we are interested in patents which are frequently members of such sets; and we

Patent Number	Frequency (in Top-100K Tuples)	Patent Title
4683195	0.799022	Process for amplifying, detecting, and/or-cloning nucleic acid sequences
4683202	0.713693	Process for amplifying nucleic acid sequences
4168146	0.202558	Immunoassay with test strip having antibodies bound...
4066512	0.175828	Biologically active membrane material
5939252	0.133269	Detachable-element assay device
5874216	0.133239	Indirect label assay device for detecting small molecules ...
5939307	0.103829	Strains of Escherichia coli, methods of preparing the same and use ...
4134792	0.084579	Specific binding assay with an enzyme modulator as a labeling substance
4237224	0.074189	Process for producing biologically functional molecular chimeras
4358535	0.02674	Specific DNA probes in diagnostic microbiology

Table 1: Patents with the highest frequency in the top-100K tuples

Patent #	Degree	Betweenness	RWR	PageRank
4683195	**1.000**	0.433	0.030	**1.000**
4683202	0.974	0.234	0.000	0.975
4168146	0.099	**1.000**	0.132	0.088
4066512	0.036	0.845	0.185	0.026
5939252	0.221	0.234	**1.000**	0.000
5874216	0.122	0.234	**1.000**	0.000
5939307	0.071	0.234	0.965	0.000
4134792	0.126	0.772	0.040	0.118
4237224	0.341	0.760	0.016	0.332
4358535	0.460	0.670	0.064	0.446

Table 2: Centrality scores for the 10 highest-frequency patents in the top-100K tuples

want to find which centrality measures correspond to these frequent nodes in the citation network. We generated a citation network with 23,990 nodes and 67,484 links; then calculated four centrality metrics on each node: degree, betweenness centrality, random walk with restart score (RWR), and PageRank. In the Top-K framework, there are 4 vectors: one for each centrality measure. The indices into the vectors are patents. The entry *i* in vector *j* is the (normalized) centrality score *j* for patent *i*. Table 1 lists the patents with the highest frequency in the top-100K tuples. As expected, the patents associated with DNA sequencing have the highest frequencies. Table 2 presents the centrality scores for the patents listed in Table 1. As it can be seen, the centrality scores alone could not have found these highly impacting patents (since no patent dominates all four centrality scores).

Conclusions

We demonstrated that ranking sets of nodes in a network can be expressed in terms of the Top-K problem. Moreover, we showed that this problem can be solved efficiently using fixed memory by converting the problem into a binary search tree and applying SMA* search. We also parallelized SMA* for the Top-K problem in order to achieve super-linear speedup in a distributed-memory environment. Lastly we validated our approach through experiments on both synthetic and real data sets.

References

[1] S. Russell. Efficient memory-bounded search methods. In *ECAI*, pages 1–5, 1992.