7 Top-k Queries on Web Sources and Structured Data

7.1 Computational Model for Top-k Queries over Web Sources

Typical example:
Address = „2590 Broadway“ and Price = $ 25 and Rating = 30
issued against mapquest.com, nytoday.com, zagat.com

Major complication:
some sources do not allow sorted access

Major opportunity:
sources can be accessed in parallel

→ extension/generalization of TA

distinguish S-sources, R-sources, SR-sources

Source-Type-Aware TA

For each R-source $S_i \in S_{m+1} .. S_{m+r}$ set $high_i$:= 1
Scan SR- or S-sources S_1 .. S_m
Choose SR- or S-source S_i for next sorted access
for object d retrieved from SR- or S-source L_i do {
E(d) := E(d) ∪ {i}; $high_i$:= si(q,d);
bestscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), $high_i$ for i ∉E(d);
worstscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), 0 for i ∉E(d); };
Choose SR- or R-source Si for next random access
for object d retrieved from SR- or R-source L_i do {
E(d) := E(d) ∪ {i};
bestscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), $high_i$ for i ∉E(d);
worstscore(d) := aggr{x1, ..., xm) with xi := si(q,d) for i∈E(d), 0 for i ∉E(d); };
current top-k := k docs with largest worstscore;
$worstmin_k$:= minimum worstscore among current top-k;
Stop when bestscore(d | d not in current top-k results) ≤ $worstmin_k$;
Return current top-k;

in contrast to Fagin‘s TA, keep complete list of candidate objects

Strategies for Choosing the Source for Next Access

<u>for next sorted acccess:</u>
Escore(Si) := expected si value for next sorted access to Si
(e.g.: $high_i$)
rank(Si) := w_i * Escore(Si) / c_s(Si) // w_i is weight of Si in aggr
choose SR- or S-source with highest rank(Si)

<u>for next random acccess (probe):</u>
Escore(Si) := expected si value for next random access to Si
(e.g.: ($high_i$ – low_i) / 2)
rank(Si) := w_i * Escore(Si) / c_r(Si)
choose SR- or S-source with highest rank(Si)

The Upper Strategy for Choosing Next Object and Source (Marian et al.: TODS 2004)

idea: eagerly prove that candidate objects cannot qualify for top-k

<u>for next random acccess:</u>
among all objects with E(d)≠∅ and R(d) ≠∅
choose d‘ with the highest bestscore(d‘);
if bestscore(d‘) < bestscore(d‘‘) for object d‘‘ with E(d‘‘)=∅ then
perform sorted access next (i.e., don‘t probe d‘)
else {
Δ := bestscore(d‘) – worstmin-k;
if Δ > 0 then {
consider Si as „redundant“ for d‘ if for all Y ⊆ R(d‘) – {Si}
$\sum_{j\in Y} w_j * high_j + w_i * high_i \geq \Delta \Rightarrow \sum_{j\in Y} w_j * high_j \geq \Delta$;
choose „non-redundant“ source with highest rank(Si) }
else choose source with lowest c_r(Si);
};

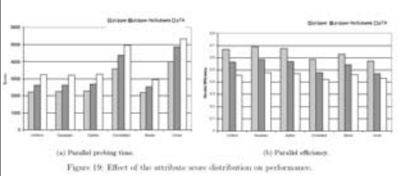
Alternative for early stage of algorithm:
could prioritize random access for objects in current top-k to improve worstmin-k

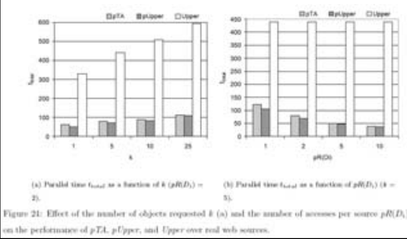
The Parallel Strategy pUpper (Marian et al.: TODS 2004)

idea: consider up to MPL(Si) parallel probes to the same R-source Si
choose objects to be probed based on
bestscore reduction and expected response time

<u>for next random acccess:</u>
probe-candidates := m objects d with E(d)≠∅ and R(d) ≠∅
such that d is among the m highest values of bestscore(d);
for each object d in probe-candidates do {
Δ := bestscore(d) – worstmin-k;
if Δ > 0 then {
choose subset Y(d) ⊆ R(d) such that $\sum_{j\in Y} w_j * high_j \geq \Delta$
and expected response time
$\sum_{Sj\in Y(d)}$ (|{d‘|bestscore(d‘)>bestscore(d) and Y(d)∩Y(d‘)≠∅}|
* c_R(Sj) / MPL(Sj))
is minimum };
};
enqueue probe(d) to queue(Si) for all Si∈Y(d)
with expected response time as priority;

Experimental Evaluation





pTA:
parallelized TA
(with asynchronous probes, but same probe order as TA)

synthetic data

real Web sources
SR: superpages (Verizon yellow pages)
R: subwaynavigator
R: mapquest
R: altavista
R: zagat
R: nytoday

from: A. Marian et al., TODS 2004

7.2 Ranking of SQL Query Results

motivated by applications such as:
realtor databases (e.g., homeadvisor.msn.com) with attributes
City, Datebuilt, Price, Sqft, Bedrooms, Bathrooms,
Deck, Fenced, Culdesac, Pool, Spa, ...

SQL queries such as
Select * From Homes Where City In (‚Redmond', ‚Bellevue') And
Bedrooms >= 3 And Bathrooms >= 2 And Price < 300000
often return too many answers or empty answers

Similar examples are car sales databases, customer support data, etc.

Straightforward idea:
apply Fagin's TA algorithm, treating indexed attributes as SR-sources and non-indexed attributes as R-sources, implemented in SQL

Less obvious issue:
similarity measures on numerical and categorical attributes

IDF Based Similarities

for query condition Ai=qi with categorical attribute Ai
the score of tuple t with t.Ai=tj is:
si(tj,qi) := idf(tj) for qi=tj, 0 else
with idf(tj) := log (|R| / |{t|t∈R and t.Ai=tj}|)

the total score for a query over multiple categorical attributes then is:
s(t,q) := $\sum_i$ si(t.Ai,qi)

Example: q: Type=‚Convertible' And Make=‚Nissan'
ranks non-Nissan convertibles higher than Nissan standard cars, 4WDs, etc.

for (practically continuous) numerical attributes A (e.g., price) define:

$$idf(t_j) = \log\left(|R| \,/\, \sum_{t\in R} e^{-\frac{1}{2}\left(\frac{t.Ai - t_j}{h}\right)^2} \right) \qquad si(t_j, q_i) = e^{-\frac{1}{2}\left(\frac{q_i - t_j}{h}\right)^2} \cdot idf(t_j)$$

for conditions Ai In (qi_1, ..., qi_k) set si(tj, qi) := $\max_{j=1..k}$ si(tj, qi_j)

QF Based Similarities

idf can be misleading about importance
(e.g., old houses are infrequent, new houses are usually looked for)

Query frequency (qf), as reflected in the current workload, can indicate importance, too:
si(tj,qi) := qf(tj) for qi=tj, 0 else

Workloads can also reveal correlations in user preferences
(e.g., frequent queries with Make In (‚Honda Accord', ‚Toyota Camry')
or City In (‚Redmond', ‚Bellevue', ‚Kirkland'))

W(t): set of previous queries with In predicate containing value t

For query value q and tuple with value t define:
si(q,t) := J(W(q),W(t))*qf(t)
with Jaccard coefficient $J(W(q),W(t)) = \frac{|W(q) \cap W(t)|}{|W(q) \cup W(t)|}$

Experimental Evaluation

4000 tuples about homes for sale in the Seattle Eastside area
80 queries each with 2-5 attributes specified
human relevance assessment by 5 colleagues

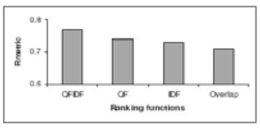


Figure 3: Quality of various ranking functions on Realtor database

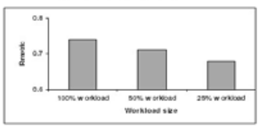


Figure 4: Ranking quality of QF Similarity on Realtor database as workload size varies

Ranking quality metric:
weighted precision R
for top n (n=10)

$$R = \sum_{i=1..n} r_i / 2^{(i-1)/n}$$

with r_i=1 if rank is relevant, 0 otherwise

from: S. Agrawal et al., CIDR 2003

Literature

- Amelie Marian, Nicolas Bruno, Luis Gravano: Evaluating Top-k Queries over Web-Accessible Databases, to appear in ACM TODS 2004
- Nicolas Bruno, Luis Gravano, Amélie Marian: Evaluating Top-k Queries over Web-Accessible Databases, ICDE Conf. 2002
- Ronald Fagin, Amnon Lotem, Moni Naor: Optimal Aggregation Algorithms for Middleware, Journal of Computer and System Science Vol. 66, 2003
- Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, Aristides Gionis: Automated Ranking of Database Query Results, CIDR Conf. 2003
- Panayiotis Tsaparas, Themistoklis Palpanas, Yannis Kotidis, Nick Koudas, Divesh Srivastava: Ranked Join Indices, ICDE Conf. 2003