SRI@work: Efficient and Effective Routing Strategies in a PDMS *

Federica Mandreoli[1], Riccardo Martoglia[1], Wilma Penzo[2], Simona Sassatelli[1], and Giorgio Villani[1]

[1] DII - University of Modena e Reggio Emilia, Italy
{federica.mandreoli, riccardo.martoglia, simona.sassatelli, giorgio.villani}@unimo.it
[2] DEIS - University of Bologna, Italy
wpenzo@deis.unibo.it

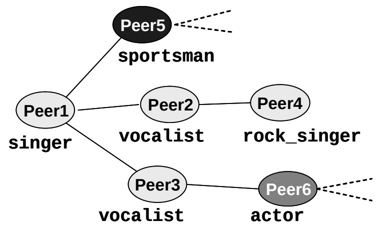
Abstract. In recent years, information sharing has gained much benefit by the large diffusion of distributed computing, namely through P2P systems and, in line with the Semantic Web vision, through Peer Data Management Systems (PDMSs). In a PDMS scenario one of the most difficult challenges is query routing, i.e. the capability of selecting small subsets of semantically relevant peers to forward a query to.
In this paper, we put the *Semantic Routing Index (SRI)* distributed mechanism we proposed in [6] at work. In particular, we present general SRI-based query execution models, designed around different performance priorities and minimizing the information spanning over the network. Starting from these models, we devise several SRI-enabled routing policies, characterized by different effectiveness and efficiency targets, and we deeply test them in ad-hoc PDMS simulation environments.

1 Introduction

In recent years, information sharing has gained much benefit by the large diffusion of distributed computing, namely through the flourishing of Peer-to-Peer systems [1]. On the other hand, in line with the Semantic Web vision, the stronger and stronger need of adding semantic value to the data has emerged. In this view, Peer Data Management Systems (PDMSs) have been introduced as a solution to the problem of large-scale sharing of semantically rich data [5]. In a PDMS, each peer is enriched with a schema (involving a set of concepts and relations among them) representing the peer's domain of interests, and semantic mappings are locally established between peer's schemas. In order to query a peer, its own schema is used for query formulation, and query answers can come from any peer in the network that is connected through a semantic path of mappings [10].

In such a distributed scenario a key challenge is query routing, i.e. the capability of selecting a small subset of relevant peers to forward a query to. Let us consider, for instance, the sample scenario of a portion of network represented in Figure1(a). It is a PDMS organized in Semantic Overlay Networks (SONs) [4]

* This work is partially supported by the FIRB NeP4B Italian project.



(a) The network and concept mappings

SRI_{Peer1}	singer	cd	title	...
Peer1	1.0	1.0	1.0	...
Peer2	0.8	0.6	0.7	...
Peer3	0.6	0.4	0.5	...
Peer5	0.1	0.2	0.3	

(b) A portion of the Peer1's SRI

Fig. 1. The reference example

where peers of the same color belong to the same SON. In the SON paradigm, nodes with semantically similar contents are clustered together in order to improve data retrieval performances. Let us suppose that a user asks Peer1 for the concept "singer". Figure 1(a) shows an example of a plausible set of pairwise and one-to-one mappings which can be defined from the queried concept (e.g. "singer" is mapped to the Peer2's concept "vocalist" which, in turn, is mapped to the Peer4's concept "rock_singer"). In such a scenario, devising routing policies which are able to exploit the semantic closeness among concepts in order to find the best paths for query forwarding is a fundamental issue. Indeed, notice that a random mechanism could propagate the query to Peer5 and thus ask for "sportsman" which is semantically dissimilar from "singer". On the other hand, a routing strategy limiting itself only to the neighboring peer mappings would prove inadequate in many cases. For instance, though "vocalist" is the corresponding concept for both Peer2 and Peer3 which thus appear equivalent for query forwarding, the path originating at Peer2 is indeed much better, since Peer3's subnetwork includes the Peer6's concept "actor". Therefore, some kind of information about the relevance of the whole semantic paths should be available in the network, maintained up-to-date, and easily accessible for query routing purposes. To this end, in [6], [7] we proposed a PDMS architecture where each peer maintains a *Semantic Routing Index (SRI)* which summarizes, for each concept of its schema, the semantic approximation "skills" of the subnetworks reachable from its neighbors, and thus gives a hint of the relevance of the data which can be reached in each path. A portion of the Peer1's SRI is shown in Figure 1(b).

With SRI at the PDMS's disposal, different routing strategies may be considered, each having effectiveness, efficiency or a trade-off between the two as its priority. For instance, coming back to our example, once the query has reached Peer2, two choices are possible: (a) Going on towards Peer4 which is the nearest unvisited peer (*efficient* strategy, the main objective being to minimize the query path); (b) navigating the way to Peer3, as it provides information more related to the queried concept (*effective* strategy, maximizing the relevance of the retrieved results). In this paper, we present two general SRI-based query execution models, designed around the two different performance priorities, efficiency and effectiveness. Both models are devised in a distributed manner through a protocol of message exchange, thus trying to minimize the information spanning over the network. Starting from these models, we devise several SRI-enabled routing policies (Section 3). Then, we deeply test the proposed approaches in complex PDMS simulation environments (Section 4). Finally, Section 5 relates our work to other approaches in the literature and concludes the paper.

2 Semantic Routing Indices

In our PDMS reference scenario each peer p stores local data, modelled upon a local schema S that describes its semantic contents, and it is connected through semantic mappings to some other neighboring peers p_j. Each of these semantic mappings M_{p_j} defines how to represent p's schema (S) in terms of the p_j' schema (S_j) vocabulary. In particular, it associates each concept in S to a corresponding concept in S_j according to a *score*, denoting the *degree of semantic similarity* between the two concepts. Such similarity is a number between 0 (no similarity) and 1 (identity).

Further, each peer maintains cumulative information summarizing the semantic approximation capabilities w.r.t. its schema of the whole subnetworks rooted at each of its neighbors. Such information is kept in a local data structure which we call *Semantic Routing Index (SRI).* In particular, a peer p having n neighbors and m concepts in its schema stores an SRI structured as a matrix with m columns and $n+1$ rows, where the first row refers to the knowledge on the local schema of peer p. Each entry $SRI[i][j]$ of this matrix contains a score expressing how the j-th concept is semantically approximated by the subnetwork rooted at the i-th neighbor, i.e. by each semantic path of mappings originated at p_j. For instance, going back to Figure 1, the score 0.8 in the Peer2 row and the "singer" column is the outcome of the aggregation of the scores associated to the paths Peer2 and Peer2-Peer4.

A formal definition of semantic mappings and SRIs can be found in [6], where the fuzzy theoretical framework of our work is presented.

3 SRI-based query processing and routing strategies

Starting from the queried peers, the objective of any query processing mechanism is to answer requests by navigating the network until a stopping condition is reached (see Section 4 for a list of stopping conditions). A query is posed on the schema of the queried peer and is represented as a tuple $q = (id, f, sim, t)$ where: id is a unique identifier for the query; f is a formula of predicates specifying the query conditions and combined through logical connectives; sim is the semantic approximation associated with the semantic path the query is following, i.e. the accumulated approximation given by the traversal of semantic mappings between peers' schemas; t is an optional relevance threshold. When t is set, those path whose relevance sim is smaller than t are not considered for query forwarding.

3.1 Query execution models

Query execution can be performed following different strategies, according to two main families of navigation policies: The *Depth First* (*DF*) query execution model, which pursues efficiency as its objective, and the *Global* (G), or *Goal-based* model, which is designed for effectiveness. Both approaches are devised in a distributed manner through a protocol of message exchange, thus trying to minimize the information spanning over the network. For both models we experienced various routing strategies, which will be the subject of the next sections. In general, the models work in the following way: Starting from the queried node, a peer p, univocally identified as p.ID, receives query q and a list L of already visited nodes from the calling peer C through an "execute" message. Then, it performs the following steps: 1. Accesses its local repository for query results; 2. decides a neighbor $Next$ among the unqueried ones which forward the query to; 3. reformulates the query q into q_{Next} for the chosen peer, using the semantic mappings M_{Next} towards it; 4. forwards the query q_{Next} to the neighbor $Next$ and waits for a finite response from it. In order to accomplish the forwarding step, the semantic information stored at the peer's SRI is used to rank the neighbors, thus also taking into account the semantic relevance of results which are likely to be retrieved from their own subnetworks. In particular, in order to limit the query path, the DF model only performs a local choice among the p's neighbors, whereas the G model adopts a global ranking policy among the already known and unvisited peers. Once the most promising neighbor has been chosen, the query q is reformulated (through unfolding [10]) into q_{Next}. Then, q_{Next} is assigned the semantic approximation value obtained by composing the semantic approximation obtained so far, i.e. $q.sim$, and the approximation for $q.f$ given by the semantic mapping M_{Next} between the current peer and the neighbor $Next$, thus instantiating $q_{Next}.sim$. Finally, the query execution process can start backtracking, returning the control to the calling peer through a "restore" message. For this reason, NS, an array of navigation states, one for each query processed by p, is maintained at each node for query restoring. As we will see in detail, while the DF model performs backtracking only when a "blind alley" is reached, the G model constantly exploits it in order to reach back potentially distant goals.

3.2 The DF query execution model

The DF query execution model provides an SRI-oriented depth first visiting criteria. The left portion of Figure 2 shows the algorithm in detail. In particular, once a peer receives an `executeDF(`q`,`C`,`L`)` message, step 1 is performed through `localExecution(`q`,`L`)`), step 2 through lines 01-07, step 3 through `prepareNextQuery(`q`,`q_{Next}`)` and step 4 through lines 11-15. Notice that only L "surfs" the network along with the query which, at each step, is locally instantiated on the peer's dictionary.

As to peer selection (lines 02-05), the evaluation of the semantic relevance of each neighbor N is performed by combining, according to a fuzzy logic approach, the similarity values in the SRI's row associated to N, i.e. $SRI[N]$, and corresponding to each concept c in the formula $q.f$: `combine(`$q.f, SRI[N]$`)`. For instance, following our reference example in Figure 1 and dealing conjunction with the minimum, the combined score for a query on Peer1 towards Peer2 involving concepts "cd" and "title" connected through AND would be $min(0.8, 0.6) = 0.6$.

```
executeDF(q,C,L):
00   localExecution(q,L);
01   NeighborList = ∅;
02   for each neighbor N of p except C
03     relevance = combine(q.f,SRI[N]);
04     add (N,relevance) to NeighborList;
05   PQ = order NeighborList in desc
       order of relevance;
     restoreDF(id,L):
06   do
07     (Next,relevance) = pop(PQ);
08     prepareNextQuery(q,$q_{Next}$);
09   while (Next not NULL and
            (Next ∈ L OR $q_{Next}$.sim <= q.t));
10   if (Next is NULL)
11     delete NS[q.id];
12     send message restoreDF(q.id,L) to C;
13   else
14     NS[q.id] = (q,PQ,C);
15     send message executeDF($q_{Next}$,p.ID,L)
            to Next;

localExecution(q,L):
00   add p.ID to L;
01   execute q on local repository;
02   if (stopping condition is reached)
03     stop execution

prepareNextQuery(q,$q_{Next}$):
00   $q_{Next}$.id = q.id;
01   $q_{Next}$.f = reformulate(q.f,$M_{Next}$);
02   $q_{Next}$.sim = compose(q.sim,combine
                (q.f, $M_{Next}$));
03   $q_{Next}$.t = q.t;
```

```
executeG(q,C,L,GL):
00   localExecution(q,L);
01   for each neighbor N of p except C
02     relevance = fScore(compose(q.sim,
                   combine(q.f,SRI[N])));
03     add (p.ID,N.ID,relevance,true) to GL;
04     order GL in desc order of relevance;
     restoreG(id,L,GL):
05   do
06     NextG = top(GL);
07   while (NextG not NULL and NextG.active
        = true and NextG.to ∈ L);
08   if (NextG is NULL)
09     stop execution
10   else if (NextG.active = false
            OR NextG.relevance <= q.t)
11     clean(p.ID,GL,NS);
12     send message restoreG(q.id,L,GL) to C;
13   else if (NextG.from <> p.ID)
14     clean(p.ID,GL,NS);
15     NH = selectNextHop(GL);
16     send message restoreG(q.id,L,GL) to NH;
17   else
18     prepareNextQuery(q,$q_{NextG.to}$);
19     NS[q.id] = (q,C);
20     NextG.active = false;
21     send message executeG($q_{Next}$,p.ID,L,GL)
          to Next;

clean(id,GL,NS):
00   for each goal G in GL
01     if (G.from = id and G.active = true)
02       return;
03     else if (G.to = id)
04       target = G;
05   delete target from GL;
06   delete NS[id];
```

Fig. 2. Query execution algorithms for "DF" (left) and "G" (right) routing models

The peer produces a list NeighborList and such list is then ordered on the basis of the peers' relevance (producing a priority queue PQ, line 05) such that the top neighbor roots the subnetwork with the best approximation skills. L is used for cycle detection, i.e. in order to avoid querying the same peer more times (line 09). The forwarding process starts backtracking when the list of unvisited neighbors for the current peer is empty, returning the control to the calling peer (lines 10-12). When a peer receives the message `restoreDF(id,L)`, it reactivates the navigation state $NS = (q, PQ, C)$ for the query identifier $q.id$ and then follows lines 06-15 either to select the next peer which to forward the query to, or to continue backtracking. Thus, DF progresses by going deeper and deeper until the stopping condition is verified.

Based on the DF model, we devised the two following routing policies, that will be tested in depth in Section 4:

- **DF policy**: the "standard" depth-first policy, straightly implementing the DF model;
- **DFF (Depth-First Fan) policy**: a variation of DF, performing depth-first visit with an added twist. Specifically, at each node, DFF performs a "fan" by exploring all the neighbors having similarity above threshold t, then it proceeds in depth to the best subnetwork, as DF does. DFF is an attempt to enhance DF, as it tries to capture in less hops more answers coming from

short semantic paths and, thus, being potentially more relevant than those retrieved by DF.

Example 1. Let us consider our reference network and see how a query posed on Peer1 and involving the concept "singer" would be routed. We use the following notation: We present the routing sequence of hops as an ordered list, where each entry N means PeerN is accessed and queried, i.e. 3 stands for Peer3, while (N) denotes a backtracking hop through PeerN. We consider the navigation until peers 1 through 4 (the most relevant ones) have been queried. Threshold t is set to 0. For the DF policy this would be the behavior: 1, 2 (most promising subnetwork rooted at Peer1), 4, (2), (1), 3. For DFF: 1, 2, (1), 3, (a fan is performed before exploring the best subnetwork), (1), 5, (1), (2), 4.

3.3 The G query execution model

Along with the DF model, we devised an additional one sharing some common principles but having effectiveness as its primary goal: The right part of Figure 2 shows the G model in detail. Differently from the DF one, in the G model each peer chooses the best peer to forward the query to in a "global" way: It does not limit its choice among the neighbors but it considers all the peers already "discovered" (i.e. for which a navigation path leading to them has been found) during network exploration and that have still not been visited. This is mainly achieved by managing and passing along the network an additional structure, called *Goal List* (GL), which is the evolution of DF's NeighborList (and PQ) and is a global ordered list of *goals*. Each goal G contains information useful for next peer selection. In particular, it represents an arc in the network topology, starting from an already queried peer and going to a destination (and still unvisited) one, and is structured as a quadruple: (G.from, G.to, G.relevance, G.active), where G.from is the goal's starting peer, G.to is the goal's destination (a neighbor of G.from), G.relevance is the relevance used for ranking and G.active is a boolean value which is used for G's logical deletion. As we will see, inactive goals are used to reconstruct the path to an already queried peer. GL is always kept ordered on the basis of the goals' relevances, which are calculated by means of the `fScore()` function (line 02). Notice that active goals are always kept ahead inactive ones in GL's ordering. Similarly to DF, the `fScore()` argument represents the semantic relevance of the goal. However, in this case the computation is `compose(`$q.sim$`,combine(`$q.f, SRI$`[N]))`[3], since, differently from DF, goals must be globally comparable and thus the whole path originating from the querying peer must be taken into account. Further, as we will see later in detail, different `fScore()` functions can be adopted in order to favor different priorities.

Following algorithm `executeG(`q,C,L,GL`)`: Each Peer, after local execution (line 00), updates GL with the current neighbors' goals (lines 01-03), orders GL (line 04) and simply selects the next goal by accessing the top goal (line 06). As in DF, L is used to avoid cycles (line 07). Then, if NextG starts from the current

[3] `compose` is a fuzzy composition function. Possible choices are the minimum or the algebraic product; for more details see [6].

peer (this means that the destination peer is a neighbor) query forwarding is executed similarly to DF (lines 18-21). Instead, if this is not the case (lines 13-16), the query is sent back one hop at a time to the peer that inserted NextG in *GL*. Notice that goals are not directly removed from GL after being selected at line 06. Instead, they are set as inactive (line 20) (i.e. already visited) before forwarding the query to their destination peer (line 21). The path to reach a previously queried peer is obtained through the function `selectNextHop(`*`GL`*`)` (line 15), whose code will not be shown in detail for simplicity of presentation; such function exploits *GL* inactive goals' information in order to identify, hop by hop, the route backtracking to the selected goal. The traversed peers are sent message `restoreG(`*`id,L,GL`*`)`. Obviously going back to potentially distant goals (peers) has a cost in terms of efficiency but always ensures the highest possible effectiveness, since the most relevant discovered peers are always selected. Notice that, in this case, backtracking is not limited to the chain of calling peers, since the selected goal does not necessarily start from one of them.

When, during backtracking, the current peer identifies that all goals starting from itself have been navigated, it can remove unnecessary information from *NS* and *GL*. These steps are performed through function `clean(`*`id,GL,NS`*`)`, invoked by peer with identifier *p*.ID=id at lines 11 and 14. In particular, the function removes the navigation state related to q from *NS* and the goal G having G.to=p.ID, since the query will not be forwarded again to p. Finally, going back to main execution, if no more goals are available in *GL* the execution is stopped (line 08-09), while, if only inactive goals are left, final backtracking is executed (lines 10-12).

Based on the G model, we devised two routing policies, which differ on the basis of the `fScore()` function and which will be tested and compared to the DF ones in Section 4:

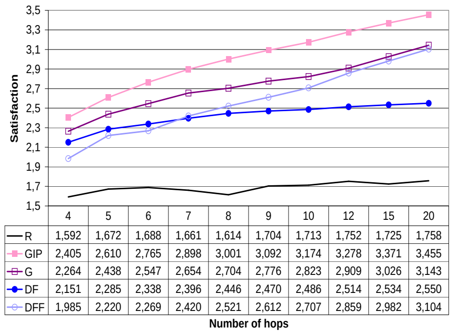
- **G policy**: The `fScore()` function is the identity and, thus, the goals are selected solely on the basis of their semantic relevance;
- **GH (Global Hybrid) policy**: This "hybrid" policy chooses goals following a trade-off between effectiveness and efficiency. This is achieved by introducing an ad-hoc parameterizable `fScore()` function, which does not only consider a goal G's semantic relevance *semRel* but also its distance *hops* (expressed in number of hops) from the current peer: `fScore`$(semRel) = semRel/(hops)^k$, $k = 0 \ldots \infty$. By simply adjusting the value of k, the GH policy can be easily tuned more on efficiency ($k \to \infty$) or on effectiveness ($k \to 0$).

Example 2. Going back to our reference example, this would be the routing sequence for the G policy: 1, 2, (1), 3 (since the relevance of the goal to Peer3 is expected to be higher than the one to Peer4), (1), (2), 4.

4 Experiments

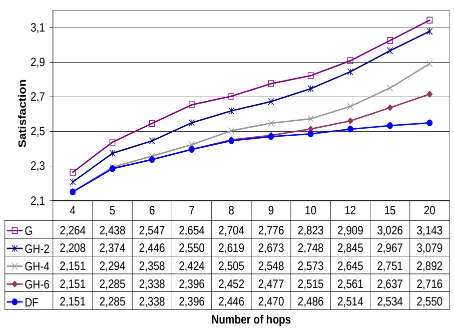
We implemented and put into practice the ideas behind our routing mechanisms in a complete infrastructure, called SUNRISE[4] [7], which completely supports

[4] System for Unified Network Routing, Indexing and Semantic Exploration



	4	5	6	7	8	9	10	12	15	20
R	1,592	1,672	1,688	1,661	1,614	1,704	1,713	1,752	1,725	1,758
GIP	2,405	2,610	2,765	2,898	3,001	3,092	3,174	3,278	3,371	3,455
G	2,264	2,438	2,547	2,654	2,704	2,776	2,823	2,909	3,026	3,143
DF	2,151	2,285	2,338	2,396	2,446	2,470	2,486	2,514	2,534	2,550
DFF	1,985	2,220	2,269	2,420	2,521	2,612	2,707	2,859	2,982	3,104

Fig. 3. Satisfaction reached by routing policies given a maximum number of hops k

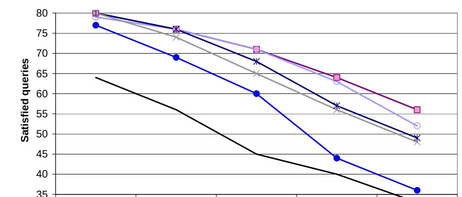


	4	5	6	7	8	9	10	12	15	20
G	2,264	2,438	2,547	2,654	2,704	2,776	2,823	2,909	3,026	3,143
GH-2	2,208	2,374	2,446	2,550	2,619	2,673	2,748	2,845	2,967	3,079
GH-4	2,151	2,294	2,358	2,424	2,505	2,548	2,573	2,645	2,751	2,892
GH-6	2,151	2,285	2,338	2,396	2,452	2,477	2,515	2,561	2,637	2,716
DF	2,151	2,285	2,338	2,396	2,446	2,470	2,486	2,514	2,534	2,550

Fig. 4. Comparison of GH policy varying w.r.t. G and DF policies

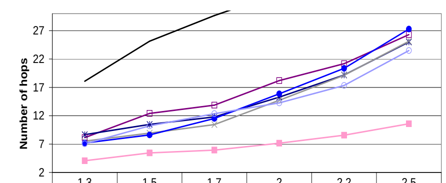
the construction of a PDMS semantic network and its testing through a dedicated *simulation environment*. By means of the SUNRISE simulation module, we were able to reproduce the main conditions characterizing such an environment without using a real P2P system. Further, we could easily abstract from network and communication issues, while maintaining full control on the internal dynamics of the network management. The simulation module is based on SimJava 2.0, a discrete, event-based, general purpose simulator. Through this framework we modelled scenarios corresponding to networks of semantic peers, each with its own schema describing a particular reality. We chose peers belonging to different semantic categories, where the schemas of the peers in the same category describe the same topic from different points of view. As in [10], the schemas are derived from real world-data sets, collected from many different available web sites, such as the DBLP Computer Society Bibliography and the ACM SIGMOD Record and enlarged with new schemas created by introducing structural and terminological variations on the original ones. Then, we distributed these schemas in the network in a clustered way, i.e. the schemas belonging to the same semantic category are located close to each other. This reflects realistic scenarios where nodes with semantically similar content are often clustered together. As to the topology of the semantic network of mappings connecting the peers, we tested our techniques on different alternatives generated with the BRITE topology generator tool. The mean size of our networks was in the order of some hundreds of nodes.

In order to evaluate the effectiveness and efficiency of our routing mechanisms, we simulated the querying process by instantiating different queries on randomly selected peers and propagating them until a stopping condition is reached. We considered two alternatives: stopping the querying process when a given number of hops (*hops*) has been performed and measuring the quality of the results (*satisfaction*) or, in a dual way, stopping when a given satisfaction is obtained and measuring the required number of hops. Satisfaction is a specifically introduced quantity that grows proportionally to the goodness of the results returned by each queried peer. In particular, it is computed as the sum of the $q.sim$ values of the traversed peers. In this section, we compare our routing strategies presented in Section 3 considering two further policies: The Random (R) one, which moves randomly in the network and corresponds to a baseline,



	1,7	2	2,2	2,5	3
R	64	56	45	40	33
GIP	80	76	71	64	56
G	80	76	71	64	56
DF	77	69	60	44	36
DFF	79	76	71	63	52
GH-2	80	76	68	57	49
GH-4	80	74	65	56	48

Fig. 5. Percentage of queries that reach a given satisfaction



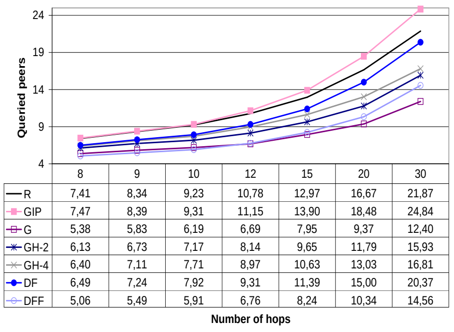
	1,3	1,5	1,7	2	2,2	2,5
R	18,09	25,15	29,66	33,59	39,11	41,75
GIP	4,07	5,45	5,95	7,15	8,57	10,58
G	8,13	12,41	13,86	18,20	21,17	26,29
GH-2	8,67	10,47	11,79	15,28	19,19	25,00
GH-4	7,51	8,94	10,45	14,72	19,14	25,14
DF	7,19	8,61	11,52	15,86	20,35	27,28
DFF	7,04	10,22	12,36	14,26	17,35	23,50

Fig. 6. Mean number of hops needed to reach a given satisfaction

and the Global IP-based (GIP) one. The GIP strategy is a variation of the Global (G) mechanism: A direct connection is established between the current peer and the peer chosen for the following step, avoiding the hops needed to reach it in the original network topology. This policy involves the modification of the PDMS network (the IP addresses of the visited peers are stored and new connections are created when necessary) and it can not be considered a real P2P strategy, but it is an interesting upper-bound to be shown. Notice that all the results we present are computed as a mean on several hundreds of query executions on several networks.

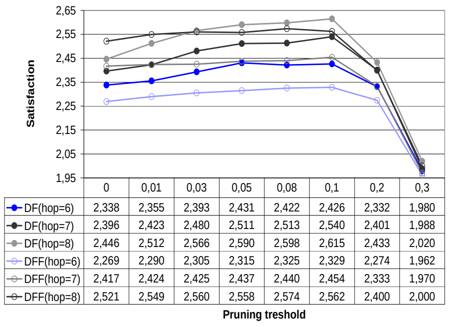
4.1 Effectiveness and Efficiency Evaluation

We start by studying the effectiveness of the routing strategies with two types of experiments. For the first one, Figure 3 shows the trend of the obtained satisfaction when we gradually vary the stopping condition on hops. As we expected, the Random strategy is outdistanced by the others and the GIP one represents the upper-bound, since all the available hops are exploited for querying peers (there are no backtracking hops). Among the others, the G policy shows the best behavior as it selects for each step the available peer with the higher sim value. Further, the DFF mechanism is initially less effective than the DF one, since it uses a large number of hops for performing its “fan” exploration. Nevertheless, for higher stopping conditions, it becomes increasingly more effective until it outperforms the DF policy and almost reaches the G one: This is due to the fact that it visits nearer peers, which have a higher probability to provide better results. The results for the GH policy are shown in Figure 4, where the trends for different values of the k parameter are represented. Notice that all the curves for the GH strategy are included in the area delimited between the G and the DF ones: The G and DF policies are indeed two particular cases of the GH one, where only the relevance (for G) or hops (for DF) component is considered, i.e. k in the formula is set to 0 (for G) or ∞ (for DF). Further, increasing the k value, the obtained behavior gets closer to the G one, as more importance is given to the relevance component. The results for the second type of experiments are represented in Figure 5, where the percentage of satisfied queries (i.e the queries for which the level of satisfaction given as stopping condition is reached) is shown



Number of hops	8	9	10	12	15	20	30
R	7,41	8,34	9,23	10,78	12,97	16,67	21,87
GIP	7,47	8,39	9,31	11,15	13,90	18,48	24,84
G	5,38	5,83	6,19	6,69	7,95	9,37	12,40
GH-2	6,13	6,73	7,17	8,14	9,65	11,79	15,93
GH-4	6,40	7,11	7,71	8,97	10,63	13,03	16,81
DF	6,49	7,24	7,92	9,31	11,39	15,00	20,37
DFF	5,06	5,49	5,91	6,76	8,24	10,34	14,56

Fig. 7. Mean number of queried peers given a maximum number of hops



Pruning treshold	0	0,01	0,03	0,05	0,08	0,1	0,2	0,3
DF(hop=6)	2,338	2,355	2,393	2,431	2,422	2,426	2,332	1,980
DF(hop=7)	2,396	2,423	2,480	2,511	2,513	2,540	2,401	1,988
DF(hop=8)	2,446	2,512	2,566	2,590	2,598	2,615	2,433	2,020
DFF(hop=6)	2,269	2,290	2,305	2,315	2,325	2,329	2,274	1,962
DFF(hop=7)	2,417	2,424	2,425	2,437	2,440	2,454	2,333	1,970
DFF(hop=8)	2,521	2,549	2,560	2,558	2,574	2,562	2,400	2,000

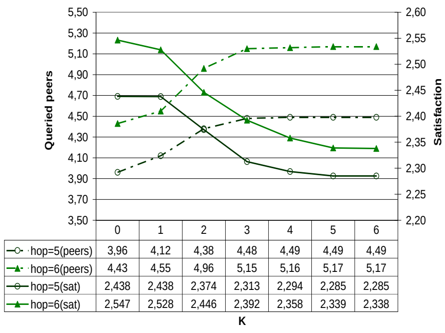
Fig. 8. Pruning threshold effect on query satisfaction w.r.t. DF and DFF policies

for different levels of satisfaction. The G policy confirms to be the most effective one, while the performances of the GH ones decrease with the value of k. As for the family of the DF strategies, the earlier explained difference between DF and DFF is even further evidenced.

The results of the experiments aiming at verifying the efficiency of the routing strategies are shown in Figures 6 and 7. Figure 6 is the dual perspective of the situation in Figure 3, since it represents the trend of the number of required hops for a given satisfaction goal. As in the previous graph, the R and GIP strategies act as the lower and upper bounds respectively. The G policy is instead the less efficient one, since at each step it chooses the next peer to visit without considering the number of hops needed to reach it. As for the GH strategies, obviously we have that the higher is k the more efficient is the routing policy. It is interesting to note that also from this efficiency perspective, after the initial phase, the DFF policy outperforms the DF one.

Figure 7 shows another measure of the efficiency corresponding to the number of queried peers for a given number of hops: Given a limit of hops, we have in fact that the more efficient is a policy the lower is the number of useless hops executed and, thus, the higher is the number of queried peers. Also in this case the less efficient policy is the G one, while for the GH family higher values of k led to a higher efficiency. The bad performances of the DFF strategy is due to the fact that, during "fan" explorations, it uses two hops for each queried peer (one for reaching it and one for going back), whereas the DF is the best policy (apart from the bound cases of R and GIP mechanisms) because each hop is used for querying the neighbor with the highest sim value.

Figure 8 explores the behavior of DF and DFF when a pruning mechanism involving a non-null $q.t$ threshold is activated. The graph shows, for both routing strategies, the satisfaction reached for a given hop limit when we vary the pruning threshold. Notice that when we use a zero threshold, and consequently apply no pruning mechanism, we obtain the same results presented earlier in the section. As can be seen, increasing the threshold leads initially to an improvement of satisfaction for both strategies. However, further increases of the threshold lead to lower values of satisfaction, signifying that useful subnetworks are pruned. The value of the optimal pruning threshold obviously depends on many factors,



K	0	1	2	3	4	5	6
hop=5(peers)	3,96	4,12	4,38	4,48	4,49	4,49	4,49
hop=6(peers)	4,43	4,55	4,96	5,15	5,16	5,17	5,17
hop=5(sat)	2,438	2,438	2,374	2,313	2,294	2,285	2,285
hop=6(sat)	2,547	2,528	2,446	2,392	2,358	2,339	2,338

Fig. 9. k influence on efficiency and effectiveness in GH policy

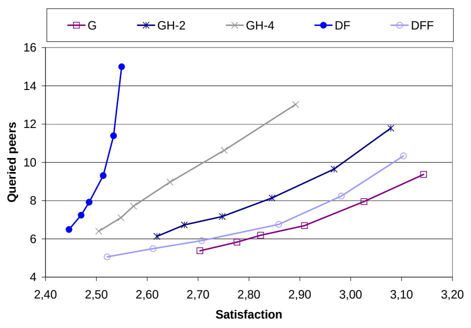


Fig. 10. Effectiveness vs. efficiency of routing policies

however we found out that setting it to 0.1 was a good choice in most tested settings, as Figure 8 shows.

The graph represented in Figure 9 deepens the study of the behavior of the GH strategies, showing the trend of satisfaction (continuous lines) and queried peers (dotted lines) for a given number of hops when we vary the value of k. As we expected, the higher the value of k, the lower is the obtained satisfaction and the higher is the number of queried peers. Increasing k we in fact give more importance to the hops component in the formula and consequently obtain more efficient but less effective strategies.

Finally, Figure 10 shows the relation between the number of queried peers and the satisfaction that every policy reaches given a maximum number of hops. In this way, we are able to visualize how the results obtained by the different policies position themselves in a combined effectiveness/efficiency plane. As expected, we observe that the G policy is the most effective one, since its curve is located near to the satisfaction axis. In contrast, the DF policy appears as the most efficient one. Moreover, we can see the effect of k in the GH policy: The increasing of k makes the GH policy more efficient, but less effective. Finally, notice that the DFF policy can reach satisfaction goals similar to the ones reached by the G strategy, but in a more efficient way.

5 Related works and concluding remarks

Much research work in the P2P area has studied query routing issues [9], [3], [2], [8], [11]. Most of these proposals are based on IR-style and machine-learning techniques [3], [2], [8], [11]. However, all of them provide routing techniques either assuming distributed indices which are indeed conceptually global [9], [8], or supporting completely decentralized search algorithms which, nevertheless, exploit information about neighboring peers only. An exception is [3] in which, however, the authors assume a common vocabulary in the network. Also in a P2P scenario, the work [12] is particularly interesting since, differently from many other works, it is completely focused on query routing policies' design and evaluation. Many of the proposed strategies are mainly designed for efficiency purposes, such as "iterative deepening" or "directed BFS", which are completely unaware of data location among peers. Instead, the "local indices" strategy makes use of distributed structures which can be configured to index, at each peer, the location of data on all surrounding peers. In this case, the indexed peers are not only limited to neighbors (an indexing radius r can be specified), however the index size is exponential on r and can not be efficiently maintained for large radii.

As we showed in [6], [7] and through the detailed experiments presented in Section 4, in order to obtain good routing effectiveness without losing network dynamism in difficultly maintainable indexing structures, some kind of summarized information about the data available in the neighbors' rooted subnetworks should be available: Our SRI framework serves this purpose and the SRI size is, for each peer, linear on the number of neighbors. Further, w.r.t. the cited existing approaches, it is also fundamental to note that querying a PDMS is very different than querying a P2P system: While P2P systems only provide IR-style keyword search and data indexing, in a PDMS the presence of heterogeneous schemas at the peers makes the problem of query routing even harder. With our SRI-enabled query routing policies, we aimed to support query routing in a PDMS: One key difference with many other approaches is that we enable a schema-based rather than a key(word)-based search, ranking (subnetworks of) peers according to the *semantic similarity* occurring between concepts in the peers' schemas. Moreover, the presented query processing models minimize the information spanning over the network and enable a variety of routing policies which can be easily selected and tuned in order to obtain the desired trade-off between effectiveness and efficiency and, thus, satisfy different users' needs.

References

1. Gnutella. `http://www.gnutella.com/`.
2. B.F. Cooper. Using Information Retrieval Techniques to Route Queries in an InfoBeacons Network. In *Proc. of DBISP2P*, 2004.
3. A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *Proc. of ICDCS*, 2002.
4. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. In *Proc. of AP2PC*, 2004.
5. A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. *IEEE TKDE*, 16(7):787–798, 2004.
6. F. Mandreoli, R. Martoglia, W. Penzo, and S. Sassatelli. SRI: Exploiting Semantic Information for Effective Query Routing in a PDMS. In *Proc. of WIDM*, 2006.
7. F. Mandreoli, R. Martoglia, W. Penzo, S. Sassatelli, and G. Villani. SUNRISE: Exploring PDMS Networks with Semantic Routing Indexes. In *Proc. of ESWC*, 2007.
8. S. Michel, M. Bender, P. Triantafillou, and G. Weikum. IQN Routing: Integrating Quality and Novelty in P2P Querying and Ranking. In *Proc. of EDBT*, 2006.
9. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, 2001.
10. I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *Proc. of ACM SIGMOD*, 2004.
11. C. Tempich, S. Staab, and A. Wranik. REMINDIN': Semantic Query Routing in Peer-to-Peer Networks Based on Social Metaphors. In *Proc. of WWW*, 2004.
12. B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proc. of ICDCS*, 2002.