2. Service Description Language

Overview

2.1 Definition of SDL

2.2 Matchmaking

2.3 Search Algorithms

2.4 Results of Simulations

2.1 Definition of SDL

Service Name: This is a *verb: noun(noun)* expression describing the service.

Inputs: The user of a service will provide zero or more inputs. We also need a specification of what inputs are expected and which of these inputs are mandatory: “English” name for each input, and a semantic type for that input. For example: *Amount:* `Integer` specifies that we have an input called *Amount* of type `Integer` and *Part:* `PartName` specifies that we have an input called *Part* of type `PartName` (which could be an enumerated type).

Outputs: Each service must specify the outputs that it provides and each output is specified in the same way as an input.

Attributes: In addition, services may have attributes associated with them. Examples of such attributes include cost (for using the service), average response time for requests to that service, etc.

Definition 2.1 (Type/Type Hierarchy $(\mathcal{T}, \leq)$)
A type τ *is a set whose elements are called "values" of* τ*. The pair* $(\mathcal{T}, \leq)$ *is called a* type hierarchy *if* $\mathcal{T}$ *is a set of types and* $\leq$ *is a partial ordering on* $\mathcal{T}$*.*

Figure 2.1 provides a hierarchy associated with the three motivating examples.

Type Hierarchy

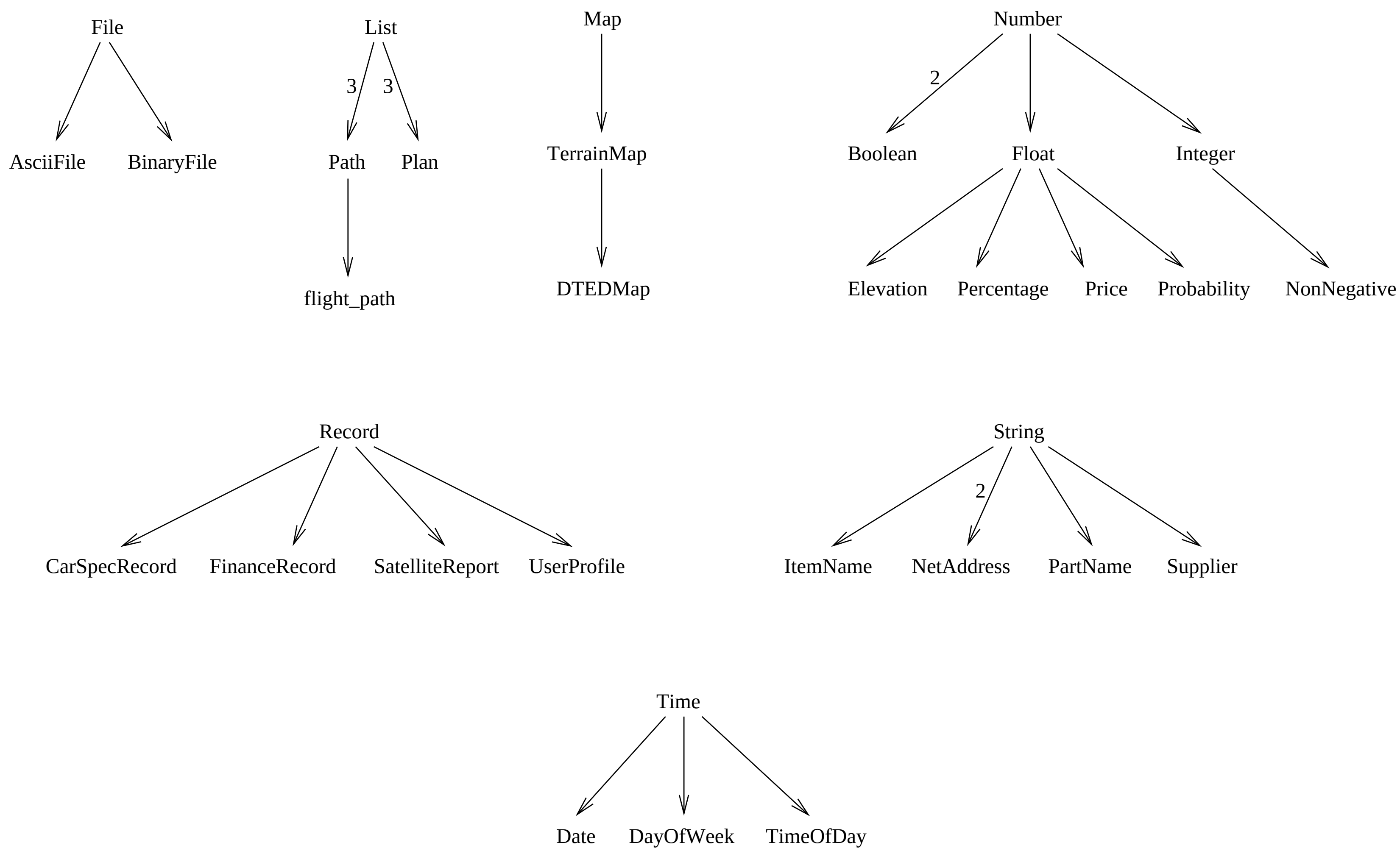


Figure 2.1: Example Type Hierarchy

Definition 2.2 (Set of Type Variables $V_{\mathcal{T}}$)
Associated with any type hierarchy $(\mathcal{T}, \leq)$, is a set $V_{\mathcal{T}}$ of symbols called type variables.

Intuitively, a type variable ranges over the values of a given type. For instance, `PartName` may be a type variable ranging over strings. When specifying the inputs required to invoke a service, we need to specify variables and their associated types. This is done in the usual way, as defined below.

Definition 2.3 (Items $s{:}\,\tau$)
If s is a variable ranging over objects of type τ, then $s{:}\,\tau$ is called an item.

$s{:}\,\tau$ may be read as saying *"the variable s may assume values drawn from the type τ"*.

Definition 2.4 (Item Atom)
If $s{:}\tau$ is an item, then $\langle\mathtt{I}\rangle s{:}\tau\langle\backslash\mathtt{I}\rangle$ (resp. $\langle\mathtt{MI}\rangle s{:}\tau\langle\backslash\mathtt{MI}\rangle$) is called an input *(resp.* mandatory input*) item atom, and $\langle\mathtt{O}\rangle s{:}\tau\langle\backslash\mathtt{O}\rangle$ is called an* output *item atom.*

Each input item is either *mandatory* or not. For example, $\langle\mathtt{MI}\rangle \mathit{Location}{:}\,\mathtt{String}\langle\backslash\mathtt{MI}\rangle$ is a mandatory input item atom, while $\langle\mathtt{I}\rangle \mathit{Nogo}{:}\,\mathtt{TerrainMap}\langle\backslash\mathtt{I}\rangle$ is a non-mandatory input item atom. The following are all valid output item atoms: $\langle\mathtt{O}\rangle \mathit{Path1}{:}\,\mathtt{Path}\langle\backslash\mathtt{O}\rangle$, $\langle\mathtt{O}\rangle \mathit{Specs}{:}\,\mathtt{CarSpecRecord}\langle\backslash\mathtt{O}\rangle$ and $\langle\mathtt{O}\rangle \mathit{Financing_plan}{:}\,\mathtt{FinanceRecord}\langle\backslash\mathtt{O}\rangle$.

Definition 2.5 (Service Description)
Let sn be a service name, $i_1,\ldots,i_n$ be input item atoms, $mi_1,\ldots,mi_k$ be mandatory input item atoms, and $o_1,\ldots,o_r$ be output item atoms. Then,

$$\begin{array}{ll} \langle \mathtt{S} \rangle & sn \\ & mi_1 \ldots mi_k \\ & i_1 \ldots i_n \\ & o_1 \ldots o_r \\ \langle \backslash \mathtt{S} \rangle & \end{array}$$

is called a service description.

Definition 2.6 (Item List)
If $s_1{:}\tau_1, \ldots, s_n{:}\tau_n$ are $n \geq 1$ items, then $\langle \mathtt{I} \rangle s_1{:}\tau_1,\ldots,s_n{:}\tau_n \langle \backslash \mathtt{I} \rangle$ is an input item list, *which is a shorthand for $\langle \mathtt{I} \rangle s_1{:}\tau_1 \langle \backslash \mathtt{I} \rangle \cdots \langle \mathtt{I} \rangle s_1{:}\tau_n \langle \backslash \mathtt{I} \rangle$; also, $\langle \mathtt{MI} \rangle s_1{:}\tau_1,\ldots,s_n{:}\tau_n \langle \backslash \mathtt{MI} \rangle$ and $\langle \mathtt{O} \rangle s_1{:}\tau_1,\ldots,s_n{:}\tau_n \langle \backslash \mathtt{O} \rangle$ are* mandatory input item lists *and* output item lists, *respectively, which are shorthands for the items $\langle \mathtt{MI} \rangle s_1{:}\tau_1 \langle \backslash \mathtt{MI} \rangle \cdots \langle \mathtt{MI} \rangle s_1{:}\tau_n \langle \backslash \mathtt{MI} \rangle$ and $\langle \mathtt{O} \rangle s_1{:}\tau_1 \langle \backslash \mathtt{O} \rangle \cdots \langle \mathtt{O} \rangle s_1{:}\tau_n \langle \backslash \mathtt{O} \rangle$, respectively.*

$\langle \texttt{S} \rangle$ *classify: user*

$\langle \texttt{MI} \rangle$*ssn:* `String` $\langle \backslash \texttt{MI} \rangle$

$\langle \texttt{I} \rangle$*name:* `String` $\langle \backslash \texttt{I} \rangle$

$\langle \texttt{O} \rangle$*class:* `UserProfile` $\langle \backslash \texttt{O} \rangle$

$\langle \backslash \texttt{S} \rangle$

This service may take as input, the name and the social security number of a user, and provide as output, a classification of the user as a "low," "medium," "high," or "very high" spender. The social security number is a mandatory input, whereas the name is optional as it can be uniquely determined from a person's social security number.

$\langle\texttt{S}\rangle$ *create: plan(flight)*

$\langle\texttt{MI}\rangle$*Location:* $\texttt{SatelliteReport}$, *Flightroute:* $\texttt{Path}$, *Nogo:* $\texttt{Map}\langle\backslash\texttt{MI}\rangle$

$\langle\texttt{O}\rangle$*Plan:* $\texttt{Plan}\langle\backslash\texttt{O}\rangle$

$\langle\backslash\texttt{S}\rangle$

This service takes three mandatory inputs (the location of the plane, the allocated flight route of the plane, and a set of Nogo areas), and generates a modified flight path for the plane.

$\langle \texttt{S} \rangle$ *monitor: availablestock*

$\langle \texttt{MI} \rangle$ *Amount:* `Integer`, *Partid:* `String` $\langle \backslash\texttt{MI} \rangle$

$\langle \texttt{I} \rangle$ *Name:* `String` $\langle \backslash\texttt{I} \rangle$

$\langle \texttt{O} \rangle$ *Status:* `String` $\langle \backslash\texttt{O} \rangle$

$\langle \backslash\texttt{S} \rangle$

This service takes the *Amount* and $Part_id$ of the requested part as mandatory inputs, and the *name* of the requested part as an optional input. The *Name* of a *part* maybe determined from its $Part_id$. This service returns as output the string `amount_available` or `amount_not_available`.

2.2 Metrik und Matchmaking

Bisher haben wir Distanzen zwischen Verben und zwischen Noun-Terms definiert. Wir wollen aber eine Distanz zwischen Service-Names.

Definition 2.7 (Composite Distance Function cd**)**
Suppose we have two different sets of words Σ_1 *and* Σ_2 *with* Σ_1*-hierarchy* $\mathcal{SH}_1 =_{def} (T_1, E_1, \wp_1)$ *and* Σ_2*-hierarchy* $\mathcal{SH}_2 =_{def} (T_2, E_2, \wp_2)$*. Let* d_1, d_2 *be the distance functions induced by* $\mathcal{SH}_1, \mathcal{SH}_2$*, respectively. Consider two pairs of words,* $\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle \in \Sigma_1 \times \Sigma_2$*. A* composite distance *function* cd *is any mapping from* $(\Sigma_1 \times \Sigma_2) \times (\Sigma_1 \times \Sigma_2)$ *to* $\mathbb{Z}^+$ *such that:*

1. $\mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_2, w'_2\rangle) = \mathrm{cd}(\langle w_2, w'_2\rangle, \langle w_1, w'_1\rangle)$ *(Symmetry)*

2. $\mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_1, w'_1\rangle) = 0$ *(Ipso-distance)*

3. *If* $d_1(w_1, w_2) \leq d_1(w_1, w_3)$, *then*
   $\mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_2, w'_2\rangle) \leq \mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_3, w'_2\rangle)$ *(Expansion of* d_1*)*

4. *If* $d_2(w'_1, w'_2) \leq d_2(w'_1, w'_3)$, *then*
   $\mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_2, w'_2\rangle) \leq \mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_2, w'_3\rangle)$ *(Expansion of* d_2*)*

5. $\mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_3, w'_3\rangle) \leq \mathrm{cd}(\langle w_1, w'_1\rangle, \langle w_2, w'_2\rangle) +$
   $\mathrm{cd}(\langle w_2, w'_2\rangle, \langle w_3, w'_3\rangle)$ *(Triangle Inequality).*

Example 2.1 (Composite Distances)
Let d_1 and d_2 be distances defined as in Section 1.2 on the verb and noun-term hierarchies given in Figure 18 on page 18 and Figure 19 on page 19, respectively. Moreover, let the composite distance function be defined as

$$\mathrm{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle) =_{def} d_1(w_1, w_2) + d_2(w_1', w_2').$$

Now consider the following two pairs: $\langle \mathit{provide}, \mathit{information} \rangle$ and $\langle \mathit{broadcast}, \mathit{data(GPS)} \rangle$. As can be seen from Figure 18 on page 18, the distance between provide and broadcast is

$$d_1(\mathit{provide}, \mathit{broadcast}) = 2,$$

as is the distance between information and data(GPS) (see Figure 19 on page 19).

Thus, the composite distance between these two pairs is given by

$$\begin{aligned}\mathrm{cd}(&\langle \mathit{provide}, \mathit{information}\rangle,\\ &\langle \mathit{broadcast}, \mathit{data(GPS)}\rangle) &=& \ d_1(\mathit{provide}, \mathit{broadcast}) + d_2(\mathit{information}, \mathit{data(GPS)})\\ &&=& \ 4.\end{aligned}$$

As another example, consider the pairs $\langle \mathit{identify}, \mathit{items}\rangle$ and $\langle \mathit{determine}, \mathit{product}\rangle$. In this case, as given by Figure 18 on page 18, the distance $d_1(\mathit{identify}, \mathit{determine})$ between *identify* and *determine* is 5. And from Figure 19 on page 19, the distance between *items* and *product* is $d_2(\mathit{items}, \mathit{product}) = 1$. Then, the composite distance between $\langle \mathit{identify}, \mathit{items}\rangle$ and $\langle \mathit{determine}, \mathit{product}\rangle$ will be the sum of their verb and nounterm distances, i.e., 6.

- Was, wenn wir die Entfernung zwischen $n_1(n_1')$ und $n_2(n_2')$ suchen aber diese Ausdr"ucke kommen nicht vor (nur n_1, n_1', n_2, n_2' sind in nt)?
  Dann benutzen wir cd wobei $\Sigma_1 := \Sigma_2 :=$ nt.
- Was wenn wir die Distanz zwischen $n_1(n_2)$ und n_1 suchen und der Term $n_1(n_2)$ kommt nicht vor? (Eventuell gibt es ein Synonym n_3 f"ur n_2 so, dass $n_1(n_3)$ vorkommt.)
  Dann sch"atzen wir grob ab: siehe Definition von $\mathbf{d_G}$.

Definition 2.8 (The Function $\mathbf{d_G}$)
We interpret, n_1 as n_1(general), e.g., information as information(general), and assume that a function denoted by $\mathrm{d_G}$ for computing the distance between any noun n and general is given to the system. The following is a possible function: Let n be a noun and let $w_1, \ldots, w_k$ be the weights of all the edges between the Noun-Term-node which includes n and any of its neighboring vertices. Then:

$$\mathrm{d_G}(n, \mathit{general}) =_{def} max(w_1, \ldots, w_k).$$

Example 2.2 (Distances)

When n is the noun-term map or navigation, $\mathrm{d_G}(n, \mathit{general}) = 2$ but when n is plan or route, $\mathrm{d_G}(n, \mathit{general}) = 1$.

Consider a query which asks for map(region). Which noun-term should we consider first?

Although there is no noun-term in our hierarchy named map(region), there are noun-terms for both map and region. Recall that $\mathrm{d_G}(\mathit{map}, \mathit{general}) = 2$. If we can find a noun-term n with a distance of 2 or less from map(region), we should start at n. Otherwise, we should start at map. In our current example, we should start at map(area) as region has a distance of 1 from area and so map(area) has a distance of $1 < 2$. However, if we were looking for map(city), there is no noun-term with a distance of 2 or less so we should start at map.

Matchmaking

We present **find_nn**: An algorithm to solve the k-nearest neighbor problem. Given a pair $\langle v, nt \rangle$ specifying a desired service, this algorithm will return a set of k agents that provide the most closely matching services.

Closeness between $\langle v, nt \rangle$ and another pair $\langle v', nt' \rangle$ is determined by using

1. the distance functions associated with the verb and noun-term hierarchies,
2. a composite distance function cd specified by the agent invoking the **find_nn** algorithm.

The algorithm also uses the AgentTable and the ServiceTable described in Subsection 1.3 on page 10, as well as the following internal data structures and/or subroutines:

Todo: List of verb/noun-term pairs (maintained in increasing order of distance), which are extended by their distances from the verb/noun-term pair that is requested in the initial or recursive call to the **find_nn** function.

ANSTABLE: Table consisting of at most k entries (k= # agents requested). While executing **find_nn**, *ANSTABLE* will contain the best answers (together with their distances) found thus far, from the requested service (v,nt).

search_service_table: This function, given a verb/noun-term pair (V,NT) and an integer K, returns the set of all agents which provide the service $V:NT)$; if their number exceeds K, it returns K of them, which are deliberatively chosen.

num_ans: This function merely keeps track of the number of answers in *ANSTABLE*.

next_nbr: This function takes as input the list $Todo$ mentioned above and a pair (V,NT). Intuitively, $Todo$ consists of verb/noun-term pairs that have not yet been relaxed. Suppose (V_1,NT_1) is the first verb/noun-term pair in this list. The *candidate-relaxation* of (V_1,NT_1), $cr(V_1,NT_1)$, is defined as follows:

1. If V' is an immediate neighbor of V_1 in the verb hierarchy, then (V',NT_1) is in $cr(V_1,NT_1)$.
2. If NT' is an immediate neighbor of NT_1 in the noun-term hierarchy, then (V_1,NT') is in $cr(V_1,NT_1)$.

The function *next_nbr* removes (V_1,NT_1) from the $Todo$ List, and then inserts all members of $cr(V_1,NT_1)$ into the $Todo$ list while maintaining the property that the $Todo$-list is in ascending order w.r.t. distance from (V,NT). The function *next_nbr* returns as output the first member of the $Todo$ list. If the $Todo$ list is empty, it returns a special pair.

relax_thesaurus: This function is called when either V or NT of the specified service name do not appear in the corresponding hierarchy. It returns a pair that is "similar" to (V,NT) whose components do appear in the hierarchies.

The function accesses the thesaurus and the verb/noun-term hierarchies. When invoked with the input pair (V,NT), it uses the following procedure:

- If V is not in the verb hierarchy, then search the thesaurus for a verb V' that is in the verb hierarchy and is synonymous with V, and set V to V'.

- If NT is not in the noun-term hierarchy, then do the following:
  - If NT is a single noun N, then search the thesaurus for a noun N' that is in the noun hierarchy and is synonymous with N. Set NT to N'.
  - If NT is a noun-term $N1(N2)$ then do the following:
    * If $N1$ is in the noun-term hierarchy let $N0 =_{def} N1$. Otherwise, search the thesaurus for a noun N' that is in the noun-term hierarchy and is synonymous with $N1$, and set $N0 =_{def} N'$. If $N0$ is still unassigned, return a special pair indicating an error.
    * Let $ntList$ be the set of all noun-terms of the form $N0(N')$ which occur in the noun-term hierarchy. If $ntList$ is the empty set, set $NT =_{def} N0$.
    * Otherwise, if $N2$ is in the noun-term hierarchy, let $adjOrigin$ (adjective origin) be $N2$. Then, search the thesaurus for a noun N' that is in the noun-term hierarchy and is synonymous with $N2$ and set $adjOrigin$ to N'. If no such N' exists, set $NT =_{def} N0$.
    * Compute $\mathrm{cd}(N0(N'), N0(adjOrigin))$ for each $N0(N')$ in $ntList$. Also, compute $\mathrm{cd}(N0, N0(adjOrigin))$ using d_G.
    * Let nt be a noun-term in $ntList$ with the least distance to $N0(adjOrigin)$. If $\mathrm{cd}(N0, N0(adjOrigin)) < \mathrm{cd}(nt, N0(adjOrigin))$, set $NT =_{def} N0$. Otherwise, set $NT =_{def} nt$.
    * If either V or NT after this step is still not in the verb or noun-term hierarchy, respectively, then return a pair indicating an error; otherwise, return the pair (V, NT).

Word	**Synonyms**
explore	inquire,examine,probe,investigate
vehicle	carrier,carriage,van,wagon,car,track
compute	approximate,calculate,deduce,estimate,evaluate
car(Japanese)	honda,nissan,mazda,toyota
scan	browse,glance,skim
car(American)	chevy, neon, dodge
title	license,entitlement,label, tag,designation
collect	acquire,bag,capture,clain,harvest,obtain
information	data,details,evidence,facts,advice,news,instruction, intelligence,knowledge,message
adjust	adapt,alter,change,deviate,modify,mutate,reform,transform
locate	discover,establish,find,place,position,situate
respond	answer,reply,acknowledge,expound,backfire,defend, retaliate
broadcast	advertise,announce,communicate,declare,disseminate, divulge,reveal,publish,report

Table 2.1: Example Thesaurus

Verb	**Noun-term**	**Agent**
broadcast	*data(GPS)*	satellite2
provide	*information(credit)*	credit3
broadcast	*data(GPS)*	satellite1
broadcast	*data*	satellite4
provide	*information*	news1
send	*data(GPS)*	satellite7
identify	*items*	contentDetermin2
ship	*freight*	truck1
provide	*data(GPS)*	satellite12
ship	*product*	shipping1
broadcast	*information*	satellite8
create	*plan(flight)*	autoPilot1
send	*data(GPS)*	satellite9
broadcast	*data(GPS)*	satellite6

Table 2.2: Example Agent Table

Algorithm 2.1 (find_nn)

find_nn(V:verb; NT:noun-term; K:integer)

(⋆ Find the K agents offering the services closest to $\langle V, NT\rangle$, and output them ⋆)
(⋆ with their distances; relax $\langle V, NT\rangle$ first, if it is not in the hierarchy. ⋆)
(⋆ Output is either ANSTABLE *or* $ERROR$ *⋆)*

$create(Todo, V, NT)$;
$ClosedList := NIL$;
$ANSTABLE := \emptyset$;
if $\langle V, NT\rangle \in \Sigma_v \times \Sigma_{nt}$ **then**
{ $done :=$ **false**;
  Sol $:= search\_service\_table(V, NT, K)$;
  while $\neg done$ **do**
  { $insert(\langle V, NT\rangle, ClosedList)$;
    $insert(\text{Sol}, ANSTABLE)$;
    $n := num\_ans(ANSTABLE)$;
    if $n \geq K$ **then** $done :=$ **true**
    else
    { $\langle V', NT'\rangle := next\_nbr(Todo)$;
      if $error(V', NT') =$ **true then** $done :=$ **true**
      else
      { $\langle V, NT\rangle := \langle V', NT'\rangle$;
        Sol $:= search\_service\_table(V, NT, K - n)$; }
else (⋆ *search thesaurus* ⋆)
{ $\langle V', NT'\rangle := relax\_thesaurus(V, NT)$;
  if $error(V', NT') =$ **true then return** $ERROR$
  else $find\_nn(V', NT', K)$; }
return $ANSTABLE$;
end.

Illustration

We consider Figures 18 and 19 and the agent- (Table 2.2 on page 45) and the thesaurus table (Table 2.1 on page 44). We choose cd as the sum of the individual distance functions.

A gps agent issues the call **find_nn**(*broadcast*, *data(GPS)*,5) to the *IMPACT* YP Server, thus limiting the output to five services.

1. $\langle V, NT\rangle$ is $\langle$*broadcast*, *data(GPS)*$\rangle$, and the list $Todo$ contains $\langle\langle$*broadcast*, *data(GPS)*$\rangle, 0\rangle$ when created.

2. Since $\langle$*broadcast*, *data(PGS)*$\rangle$ belongs to $\Sigma_v \times \Sigma_{nt}$, we execute

   search_service_table(*broadcast*, *data(PGS)*, 5).

   This assigns Sol the set $\{satellite1$, $satellite2$, $satellite6\}$, since, as can be seen from Table 2.2 on page 45, these are the agents providing the service *broadcast: data(GPS)*.

3. Thus, during the first iteration of the while loop, $\langle$*broadcast*, *data(GPS)*$\rangle$ is added to *ClosedList*, and *ANSTABLE* is updated to the set

   $$\{(satellite1,0),(satellite2,0),(satellite6,0)\};$$

   furthermore, $n := 3$ and thus $\langle$*broadcast*, *data(GPS)*$\rangle$ is relaxed by calling the function **next_nbr**, which returns $\langle$*broadcast*, *data*$\rangle$. The list *Todo* now contains the two entries $\langle\langle$*broadcast*, *data*$\rangle, 1\rangle$ and $\langle\langle$*send*, *data(GPS)*$\rangle, 1\rangle$.

4. The call **search_service_table**(*broadcast*,*data*,2) will return the singleton set $\{satellite4\}$, which provides the service *broadcast: data*. This result is inserted into *ANSTABLE* with its composite distance function value of 1.

5. As we do not have five answers yet, the **next_nbr** function is called again. This call returns the verb/noun-term pair $\langle$*send*, *data(GPS)*$\rangle$.

6. The subsequent call of **search_service_table**(*send*, *data(GPS)*,1) returns the singleton containing $satellite7$, which provides the service *send: data(GPS)*. Note that although $satellite9$ also provides the *send: data(GPS)* service, the **search_service_table** function chooses to return $satellite7$, because only one agent is requested for the result.

7. Now that there are five answers in the $ANSTABLE$, the **find_nn** function terminates by returning

$$(satellite1, 0), (satellite2, 0), (satellite6, 0), (satellite4, 1), (satellite7, 1).$$

Complexity of find_nn

Suppose $V \in \Sigma_v$ and $NT \in \Sigma_{nt}$. Also, let $\Sigma_v^V \subseteq \Sigma_v$ ($\Sigma_{nt}^{NT} \subseteq \Sigma_{nt}$) denote the set of all verbs (noun-terms) in our hierarchies whose distance from V (NT) is finite.

- In the worst case, **find_nn**(V, NT, K) will need $O(|\Sigma_v^V| \cdot |\Sigma_{nt}^{NT}| + K)$ time.
- Note however that if there are K services whose composite distance from $\langle V, NT \rangle$ is finite, then we can obtain a tighter bound. Specifically, let d_V (d_{NT}) be the maximum distance from a verb (noun-term) of one of these K services to V (NT). Furthermore, let $\Sigma_v^{d_V} \subseteq \Sigma_v^V$ ($\Sigma_{nt}^{d_{NT}} \subseteq \Sigma_{nt}^{NT}$) denote the set of all verbs (noun-terms) in our hierarchies whose distance from V (NT) is less than or equal to d_V (d_{NT}). Then in this case, **find_nn**(V, NT, K) will only need

$$O(|\Sigma_v^{d_V}| \cdot |\Sigma_{nt}^{d_{NT}}| + K) \text{ time.}$$

Range Computations

The **range** Algorithm below answers queries of the form *"Find all agents that provide a service* $vnt = \langle V', NT' \rangle$ *which is within a distance* D *of a requested service* $vnt = \langle V, NT \rangle$*"*. $Todo$ is a list of nodes to be processed, each of which is a service vnt' extended by its distance from the service vnt. The algorithm has two steps.

Step 1: **while** loop: It finds all pairs $vnt^\star = \langle V^\star, NT^\star \rangle$ that are within the specified distance D from vnt. **expand**(vnt, vnt', D) first computes the set

$$\{vnt^\sharp \mid D' =_{def} \mathrm{cd}(vnt^\sharp, vnt) \leq D, vnt^\sharp \in \mathbf{cr}(vnt'), \langle vnt^\sharp, D' \rangle \notin RelaxList\}.$$

Here, *RelaxList* contains the services which have already been considered. Then, **expand** inserts the elements of this set into $Todo$.

Step 2: Select operation on the Service Table, finding all agents that offer any of the service names identified in the first step. As in the **find_nn** algorithm, if V or NT are not in the relevant verb or noun-term hierarchies, algorithm **range** calls the **relax_thesaurus** procedure specified in the previous section to find a similar pair which belongs to them.

Algorithm 2.2 (range)
range(V:verb; NT:noun-term; D:real)

(⋆ Find all agents offering a service within distance D to $\langle V, NT \rangle$ ⋆)
(⋆ Output is either an ANSTABLE *or an $ERROR$ ⋆)*

1. **if** $D < 0$ **then return** $ERROR$;
2. **if** $\langle V, NT \rangle \in \Sigma_v \times \Sigma_{nt}$ **then**
3. { $RelaxList := \mathtt{NIL}$;
4. $Todo := \langle \langle V, NT \rangle, 0 \rangle$;
5. **while** $Todo \neq \mathtt{NIL}$ **do**
6. { $\langle \langle V', NT' \rangle, D' \rangle :=$ *first element of* $Todo$;
7. *insert* $\langle \langle V', NT' \rangle, D' \rangle$ *into* $RelaxList$;
8. *remove* $\langle \langle V', NT' \rangle, D' \rangle$ *from* $Todo$;
9. $expand(\langle V, NT \rangle, \langle V', NT' \rangle, D)$;
10. }
11. **return** $\pi_{Agents,Dist}(RelaxList[Verb = V', NounTerm = NT']ServiceTable)$
12. }
13. **else**
14. { (⋆ *search thesaurus* ⋆)
15. $\langle V', NT' \rangle := relax\_thesaurus(V, NT)$;
16. **if** $error(V', NT')$ **then return** $ERROR$
17. **else return** $range(V', NT', D - cd(\langle V, NT \rangle, \langle V', NT' \rangle))$;
18. }

end.

Complexity of range

Lemma 2.1 (Complexity of range)

Suppose $V \in \Sigma_v$ *and* $NT \in \Sigma_{nt}$. *Also, let*
$vnt = \{\langle v, nt\rangle \mid v \in \Sigma_v \wedge nt \in \Sigma_{nt} \wedge \mathrm{cd}(\langle v, nt\rangle, \langle V, NT\rangle) \leq D\}$ *and let* S *be the set of all (agent name,* $\langle v, nt\rangle$*) pairs where* $\langle v, nt\rangle \in vnt$.

Then **range**(V, NT, D) *will need* $O(|vnt| + |S|)$ *time.*

2.3 Simulations Ergebnisse

We are interested in

- the efficiency of finding *similar* services and
- the quality of the *matching* services provided as the output.

Performance Results

Based on a *NASA* hierarchy consisting of 17,445 words (for experimental purposes, the same hierarchy was used as both a verb and a noun hierarchy, although the *IMPACT* prototype uses different hierarchies). Weights on all edges in the hierarchies were assumed to be 1 and the composite distance function was taken to be `sum`. The algorithms were implemented in C++ and the experiments were conducted on a Sun Sparc.

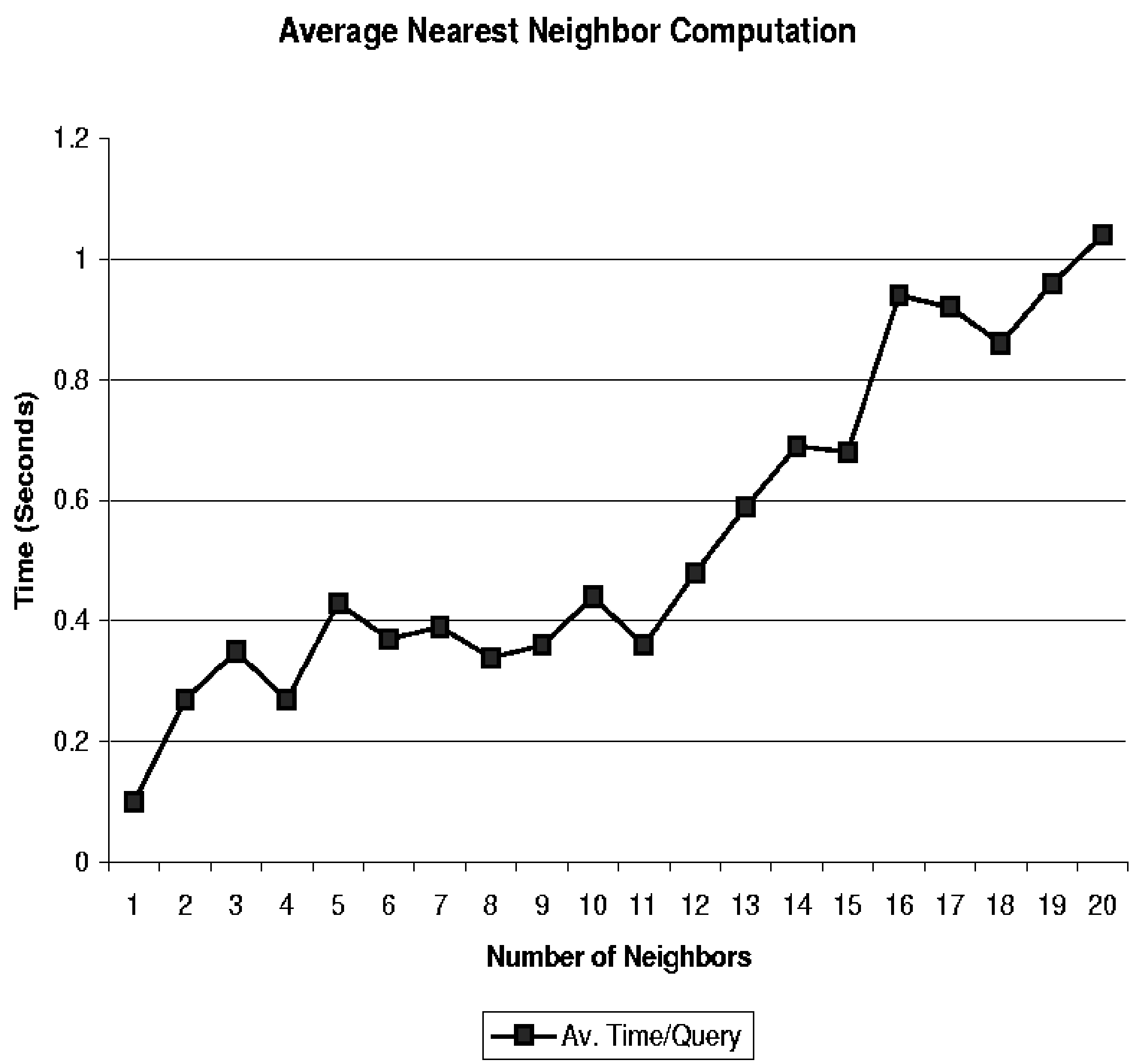


Figure 2.2: Performance of k-nearest neighbor algorithm, Average Time

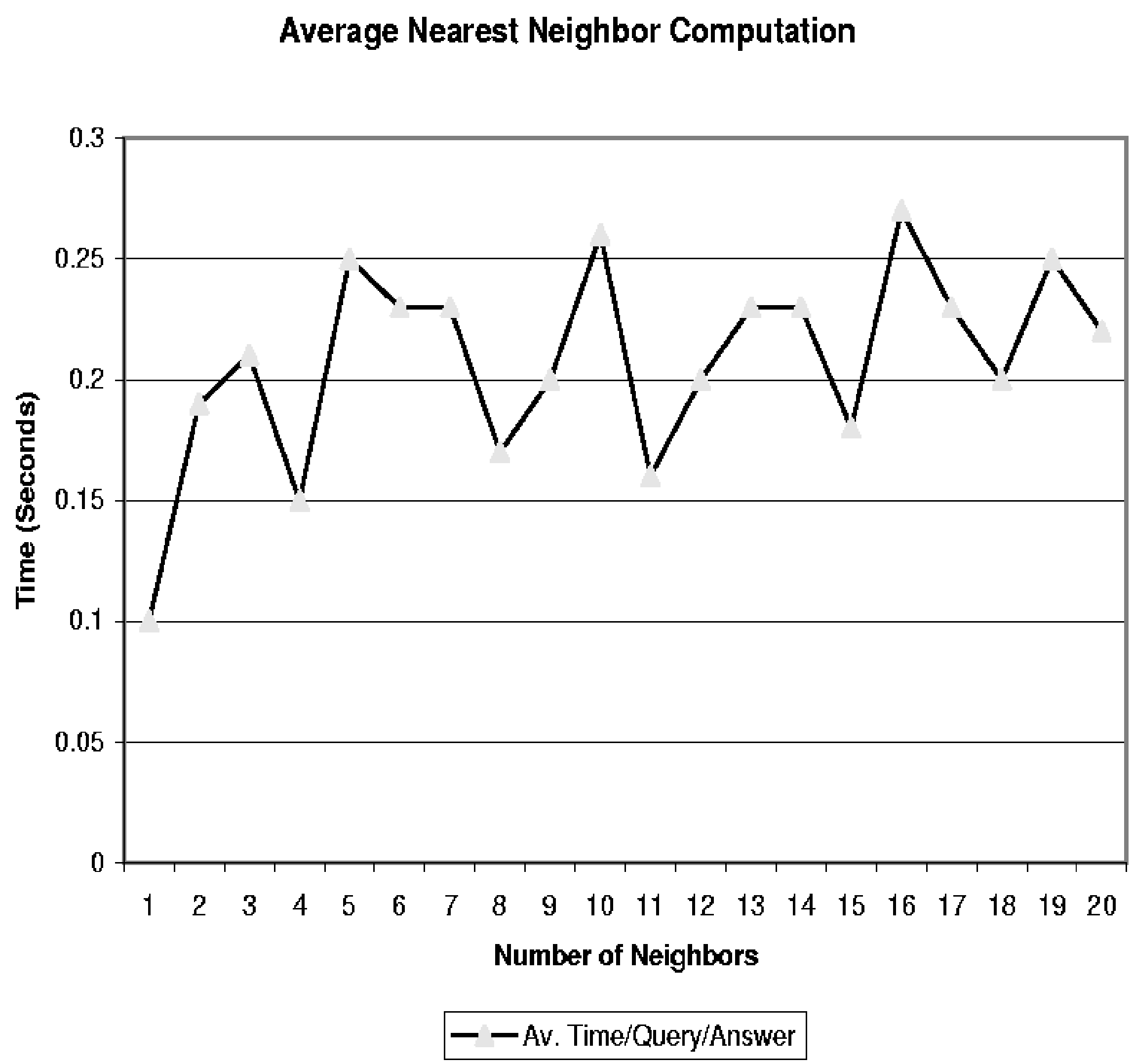


Figure 2.3: Performance of k-nearest neighbor, Average time per answer

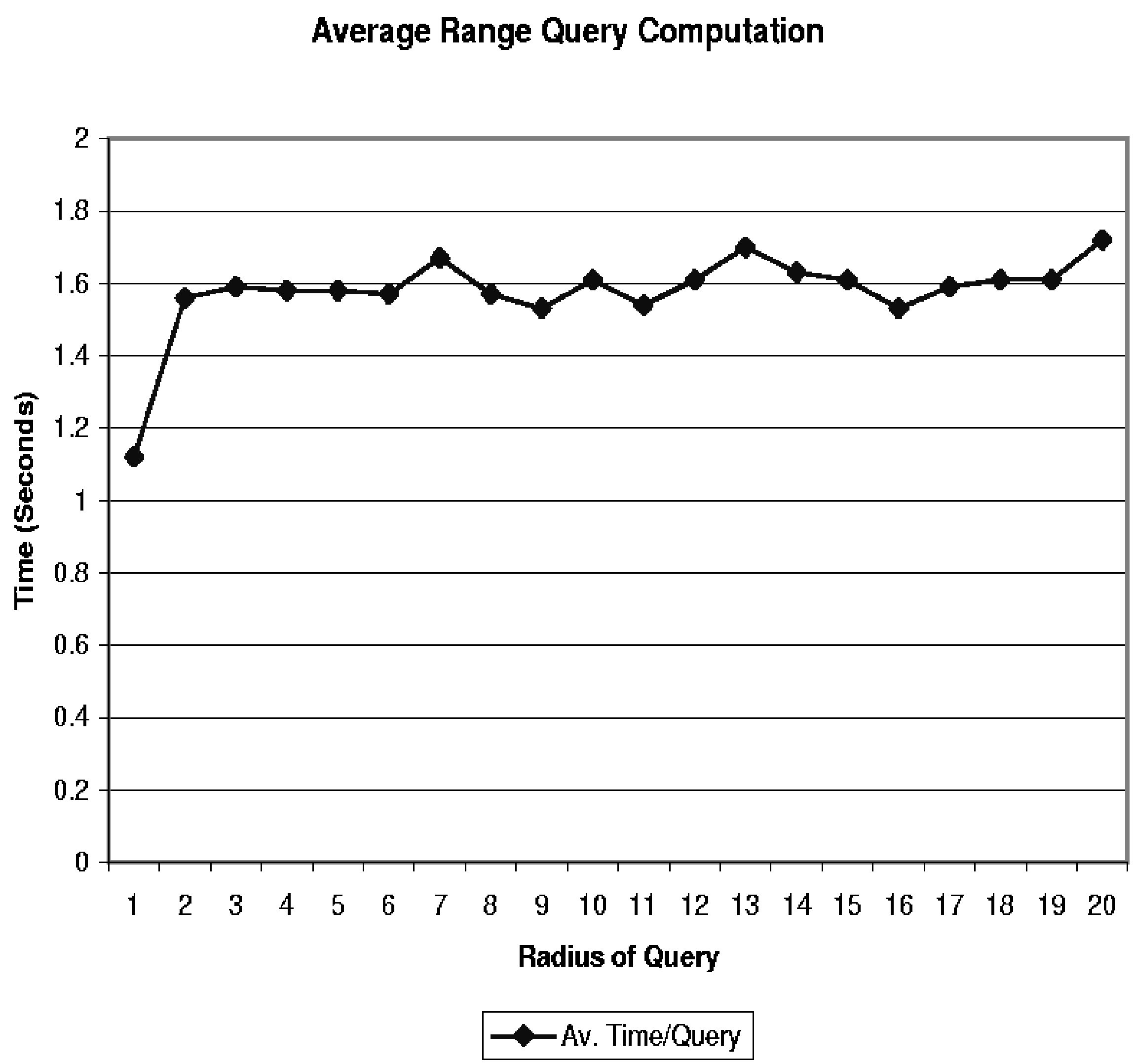


Figure 2.4: Performance of range query algorithm, Average Time

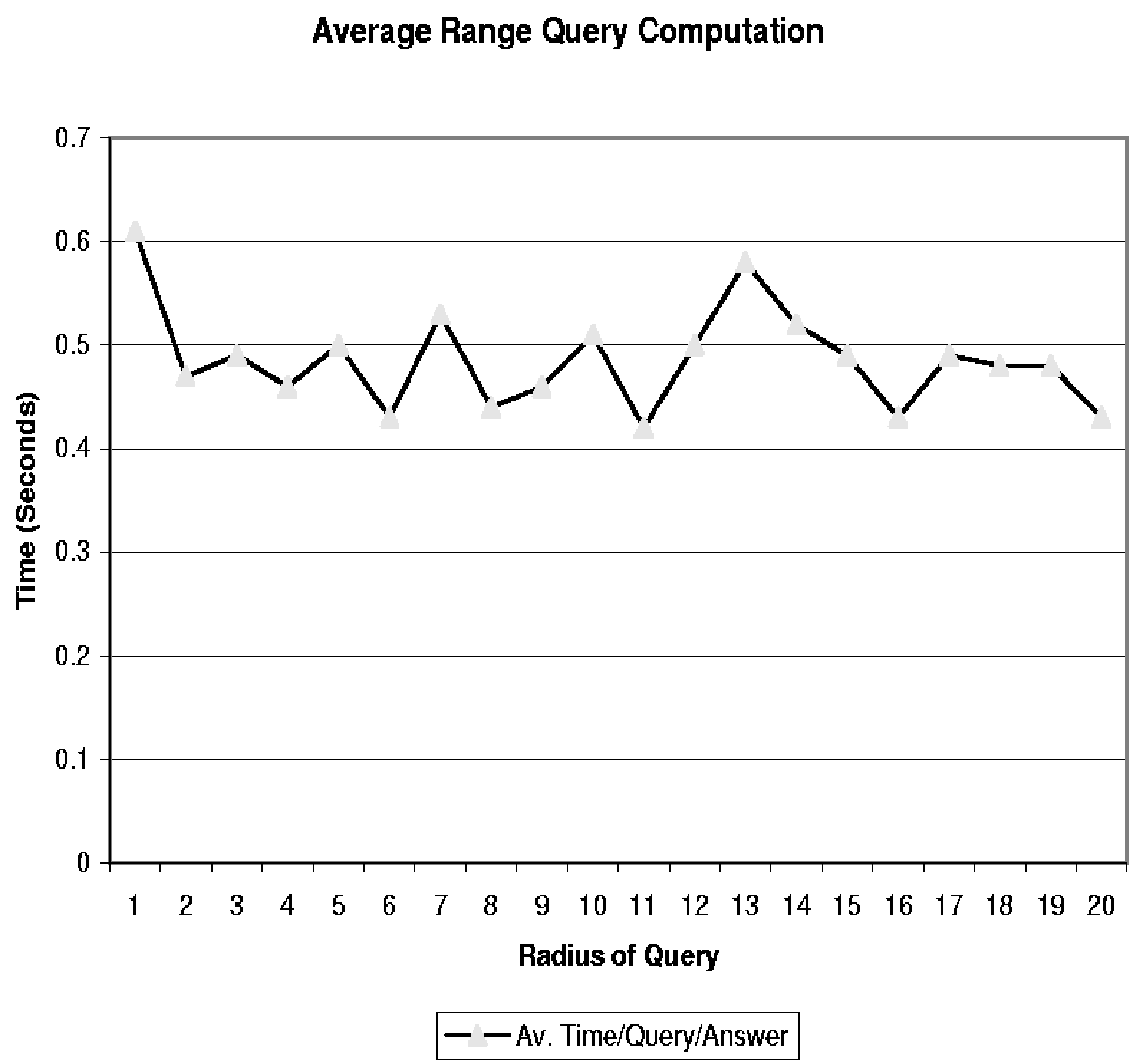
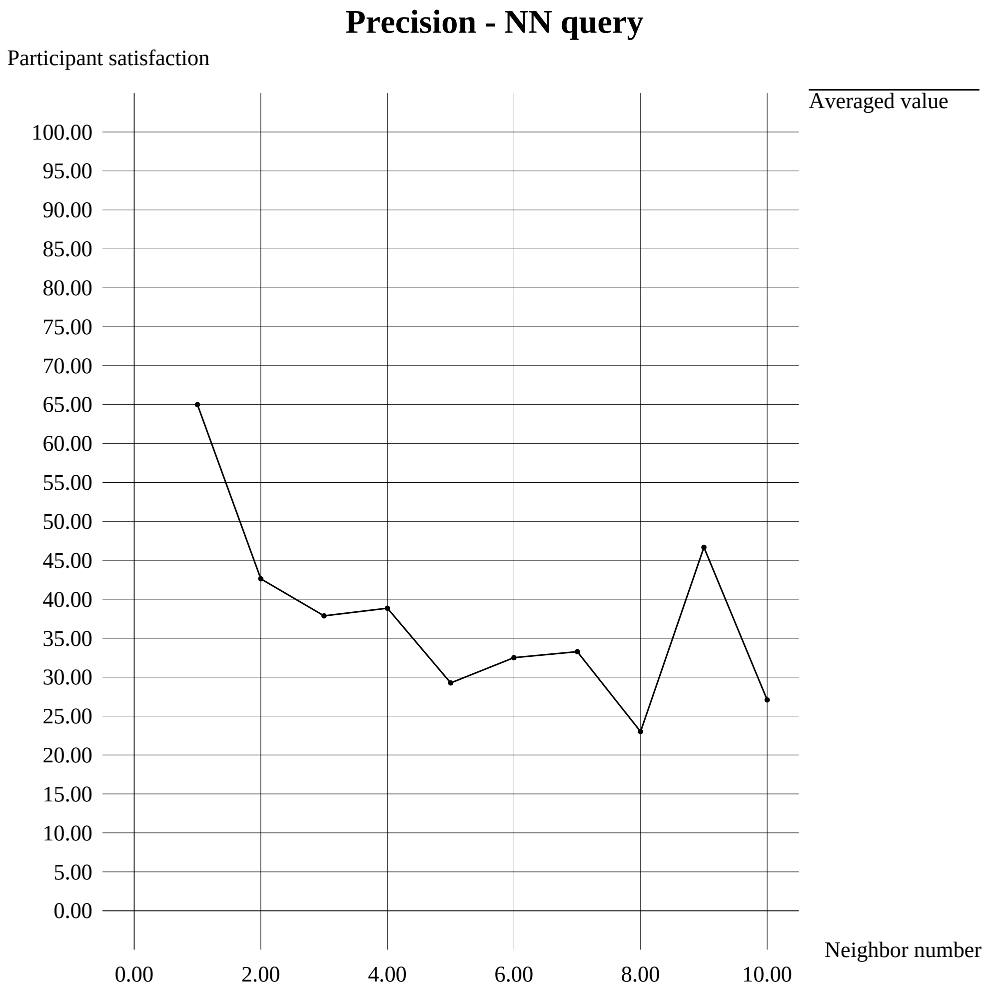


Figure 2.5: Performance of range query algorithm, Average time per answer

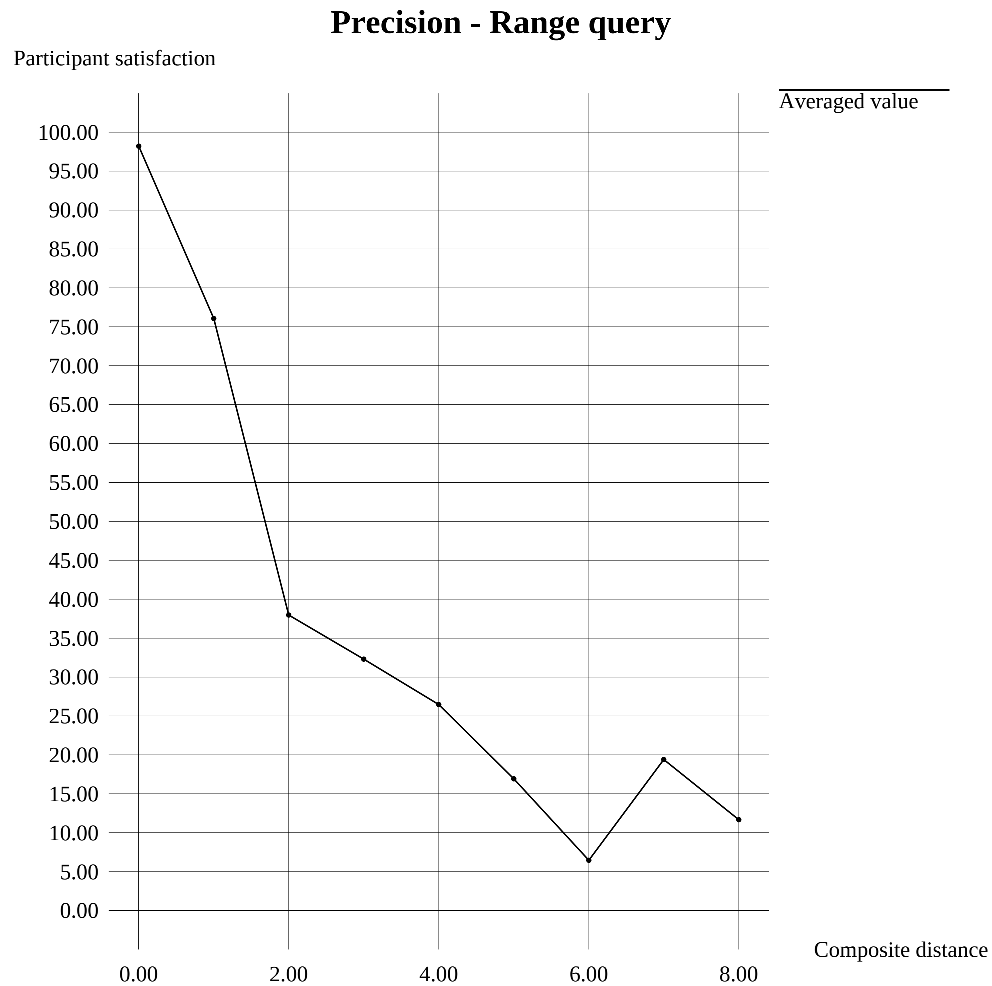
Quality of Returned Matches

We conducted an experiment involving 35 participants:

1. We used a simple verb hierarchy (10 nodes), noun-term hierarchy (90 nodes), and ServiceTable (100 services).
2. After an initial training phase, participants entered a precision phase where they were asked to perform 10 nearest neighbor and 10 range queries of their choice.
3. After each query result, participants typed in a ranking between 0 (least satisfied) and 100 (most satisfied).
4. Average satisfaction for nearest neighbor and range queries are shown below.



Precision for the k-nearest neighbor algorithm

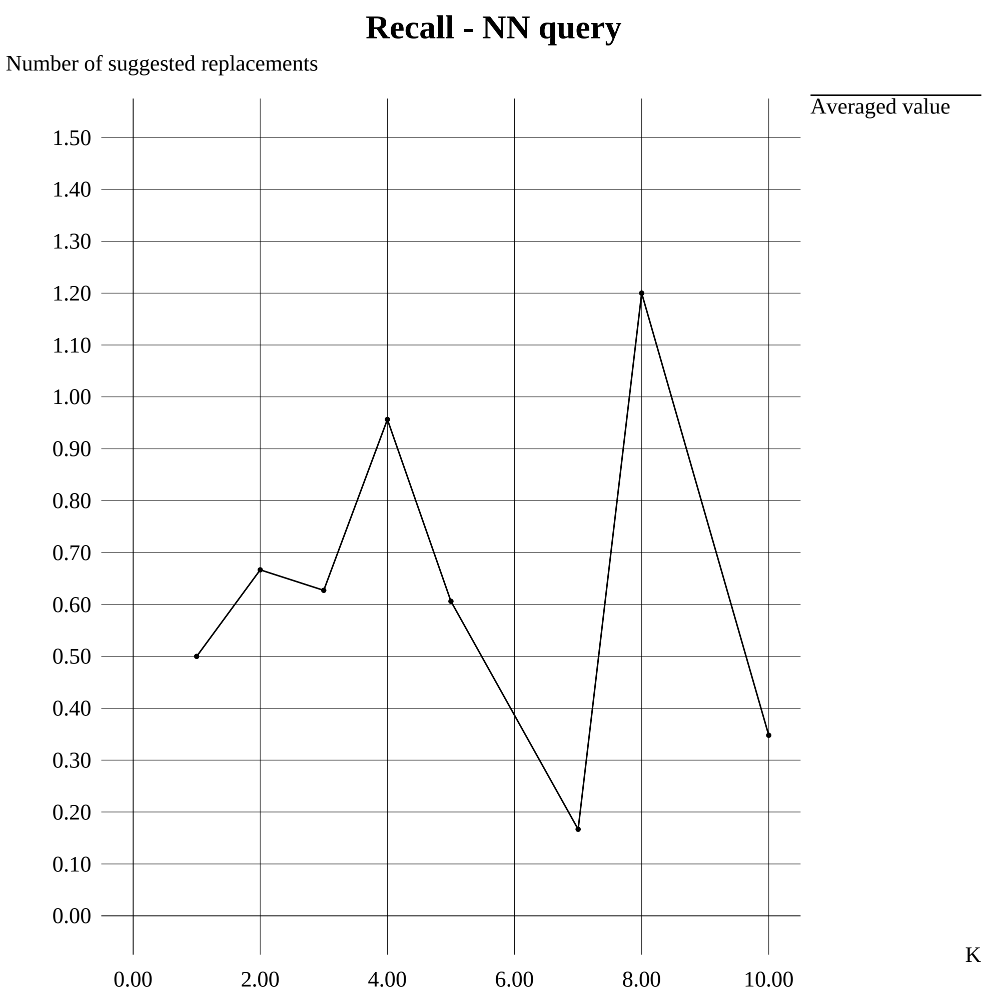


Precision for the range algorithm

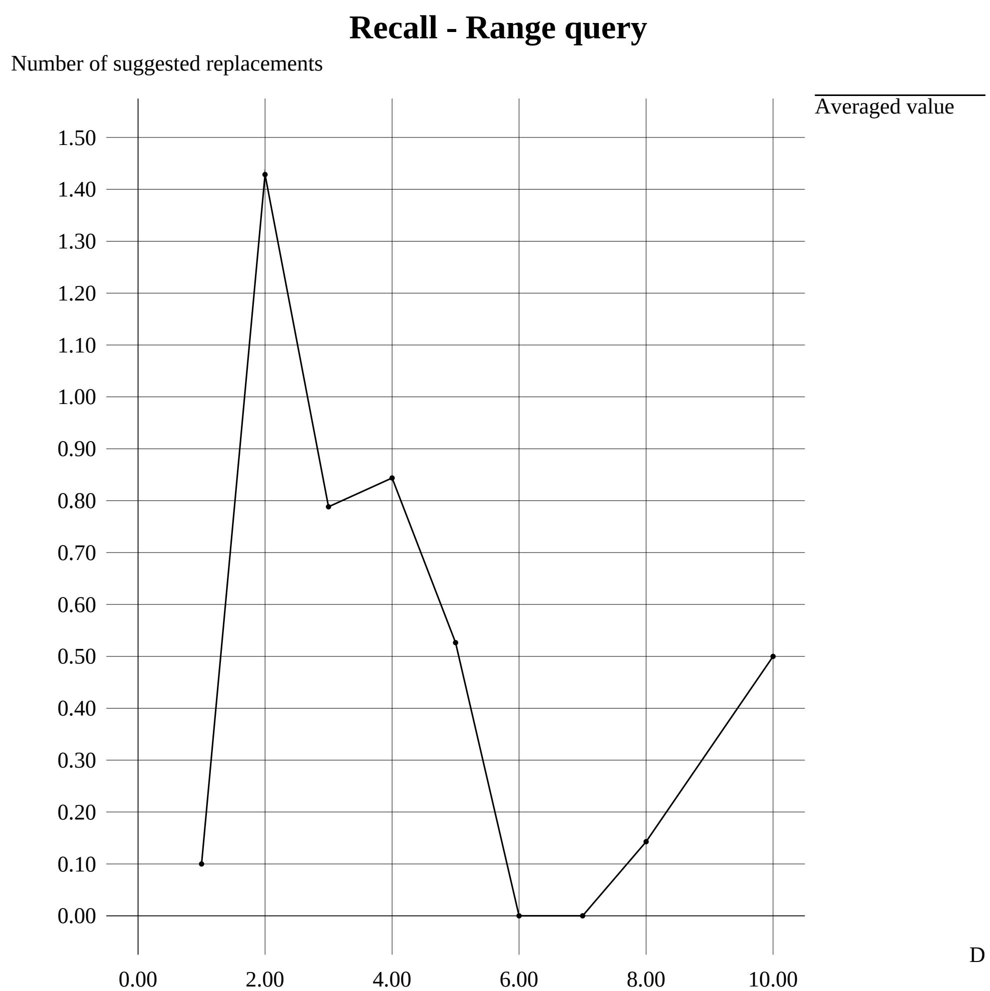
Figure 2.6: Experimental Results of Precision of our Algorithms

After completing the precision phase, participants started the recall phase.

1. They were allowed to view the ServiceTable (which up to this point was not available to them).
2. Meanwhile, they were presented with text boxes containing the query answers they gave in the previous phase.
3. After each answer, they were instructed to type in the name of all services in ServiceTable which did not appear as a query result but which should have been returned as an answer.
4. The average number of these "suggested replacements" for nearest neighbor and range query answers are shown below.



Recall for the k-nearest neighbor algorithm



Recall for the range algorithm (radius D)

Figure 2.7: Experimental Results of Recall of our Algorithms