SERGEANT: A Framework for Building More Flexible Web Agents by Exploiting a Search Engine

Shou-de Lin
Computer Science Department,
University of Southern California
4676 Admiralty Way,Suite1001,
Marina Del Rey, CA, 90292
sdlin@isi.edu
Craig A. Knoblock
Information Sciences Institution
University of Southern California
4676 Admiralty Way,Suite1001,
Marina Del Rey, CA, 90292
knoblock@isi.edu

Abstract

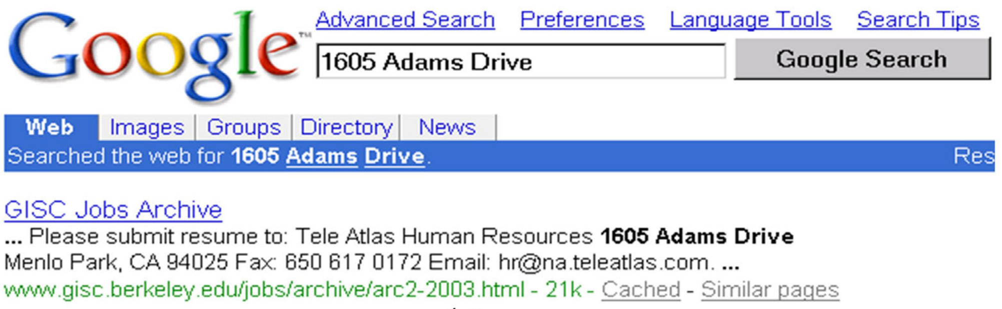
With the rapid growth of the World Wide Web, there is growing interest in developing web agents that interact with online services to acquire information. However, finding the online services perfectly suited for a given task is not always feasible. First, the agents might not be given sufficient information to fill in the required input fields for querying an online service. Second, the online service might generate only partial information. Third, the agents might need to know the information about B by some input set A, but they can only find the online services that generate A from B. Fourth, most of the online services do not tolerate errors in the inputs, thus even a minor typo in the input field can hinder them from generating any meaningful results. This paper proposes SERGEANT,[1] a framework for building flexible web agents that handle these imperfect situations. In this framework we exploit an information retrieval (IR) system as a general discovery tool to assist finding and pruning information. To demonstrate SERGEANT, we implemented two web agents: the Internet inverse geocoder and the address lookup module. Our experiments show that these agents are capable of generating high-quality results under imperfect situations.

1. Introduction

Web agents in general adopt two orthogonal strategies to gather information from the Internet. The first is to rely on an information retrieval (IR) system (e.g. a search engine as shown in Figure 1.a). Many question answering (QA) [1, 2] systems, for example, uncover answers from the search results. The second is by querying appropriate online services, e.g. a web agent that gathers geographic data usually queries an online geocoder for the latitude/longitude (lat/long) corresponding to a given address (Figure 1.b shows the input for a geocoder and Figure 1.c demonstrates the output).

[1] The key of our framework is to integrate the search engine into a web agent, and SERGEANT is pronounced as the integration of the first syllable of "search engine" with the last syllable of "web agent"

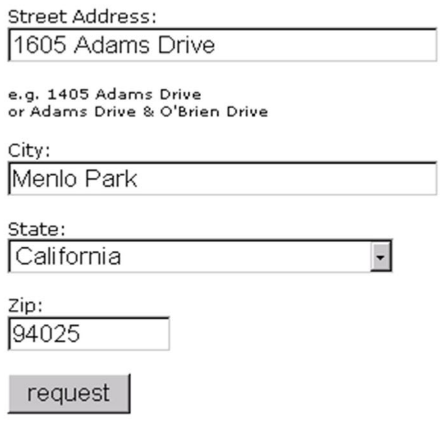
In this paper, we define an online service as an Internet service that provides an interface for the users or agents to interact with its internal program for relevant information. The tasks performed inside the internal program can be as simple as querying its local database or as complicated as integrating various information from different sites. Nonetheless, the web agents tend to view the internal functionality as a black box since the process is unknown. In general to query these online services the agent has to provide an input set $x_{1...}x_m$ and the online services will accordingly generate the output set $y_1...y_n$. For instance the geocoder site[2] is a typical online service in the sense that $x_{1...}x_m$ represents an input address while $y_1...y_n$ is the corresponding latitude and longitude.



Searched the web for **1605 Adams Drive**.

GISC Jobs Archive
... Please submit resume to: Tele Atlas Human Resources **1605 Adams Drive** Menlo Park, CA 94025 Fax: 650 617 0172 Email: hr@na.teleatlas.com. ...
www.gisc.berkeley.edu/jobs/archive/arc2-2003.html - 21k - Cached - Similar pages

1.a



Street Address: 1605 Adams Drive

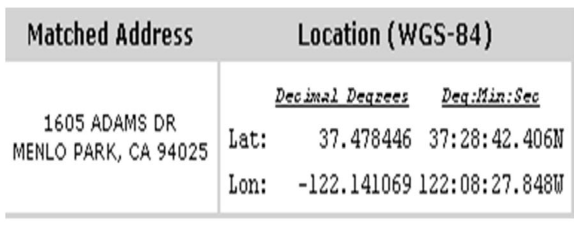
e.g. 1405 Adams Drive
or Adams Drive & O'Brien Drive

City: Menlo Park

State: California

Zip: 94025

1.b



Matched Address	Location (WGS-84)		
		Decimal Degrees	Deg:Min:Sec
1605 ADAMS DR MENLO PARK, CA 94025	Lat:	37.478446	37:28:42.406N
	Lon:	-122.141069	122:08:27.848W

1.c

Figure 1: Two strategies for web agents to acquire information. 1.a is the inputs/outputs for a search engine. 1.b shows an input to a geocoder service and 1.c demonstrates the output.

These two information-gathering strategies, utilizing the search engine and querying an online service, are diverse in many aspects. First, the information found via these two strategies is different. Search engines surf through static documents on the Internet but are incapable of interacting with online services to acquire specific information from them. For example, the web agent that utilizes the search engine usually cannot uncover the lat/long given the address. On the other hand, online services, usually designed for providing certain types of information, supply only domain-specific data and thus cannot be applied as generally as a search engine.

[2] http://geocode.com/eagle.html

Divergence also arises from the characteristics of their inputs as well as outputs. The web agent can in general adopt flexible keywords as inputs while utilizing a search engine. The inputs required for the agents to interact with online services are, on the contrary, quite restricted. The online service accepts only a certain type of data (e.g. the zip code can only be a five-digit number) and sometimes there are implicit constraints or correlation among different input fields (e.g., for city and state, there is no New York City in California). The outputs are also organized differently: a search engine outputs arbitrary documents, structured or unstructured. On the other hand the online services usually output data in a structured or semi-structured format, which in most cases can be extracted easily and precisely by a wrapper [3, 4]. Table 1 summarizes the comparison between these two strategies.

This paper describes the idea of developing a more flexible web agent by integrating these two strategies in order to exploit the strengths of each. There are two potential prospects of integration. The first is to keep using online services as the core information-discovering approach in a web agent and apply the search engine as an auxiliary tool to handle the limitations of the online sources. The second is to enhance the facility of a search engine for interacting with the online service to improve the recall rate of search results. In this paper we will focus on the first prospect.

Table 1 The comparison between online services and search engines

	Online services	*Search engines*
Information provided	Specific (deep)	General (broad)
Inputs	Restricted type of information	Flexible keywords
Outputs	Structured or semi-structured	Arbitrary documents

2. Limitations of online services

In this section we would like to address four potential limitations of online services. Ideally an intelligent web agent should have the capability to generate high quality results even if these limitations exist.

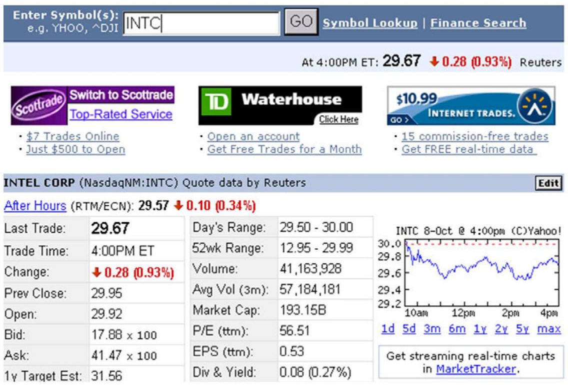
The first limitation is the existence of required inputs. Many online services require valid information as inputs, but it is not always possible to provide sufficient information to fill in all the required fields. For instance, in many sites that provide stock information such as Yahoo Finance,[3] it is required to use the "ticker symbol" (Figure 2.a) as the input instead of the company name, but sometimes the agents are given only the latter.

The second limitation is the incompleteness of the output. In some cases the online services are incapable of returning all the information the agents are looking for. The users would like, in the ideal world, an intelligent web agent that can automatically produce the missing information. For example, a web agent that utilizes the Yahoo Yellowpage site can discover a company's phone number and address information but not the ZIP code given its name (Figure 2.b). This agent, however, will not be able to satisfy the agents or users that need the zip code.

[3] http://finance.yahoo.com/

The third limitation is the lack of reversibility. The majority of online sources provide only one-way lookup services. For instance, many online email-finding services return people's email given their names but not vice versa. Resolving inverse queries given only the forward lookup service is a challenging non-deterministic task. This is theoretically solvable by exhaustive search given a finite input domain, but in practice it is usually computational intractable. Nevertheless, users would still prefer to have a web agent that performs the inverse task even if only forward services exist.

The fourth limitation is the lack of ability to tolerate minor errors in the input fields. In other words, many online services cannot return meaningful results while there is an error in the input fields. For instance, Mapquest[4] cannot generate the city map of Los Angeles if it is misspelled to "Los Angels" as the input (Figure 2.c). Ideally the users would prefer their web agent to tolerate such minor input errors and still produce plausible results.



Enter Symbol(s): e.g. YHOO, ^DJI INTC GO Symbol Lookup | Finance Search

At 4:00PM ET: 29.67 ↓0.28 (0.93%) Reuters

Switch to Scottrade Top-Rated Service
· $7 Trades Online
· Just $500 to Open

TD Waterhouse Click Here
· Open an account
· Get Free Trades for a Month

$10.99 INTERNET TRADES. GO >
· 15 commission-free trades
· Get FREE real-time data

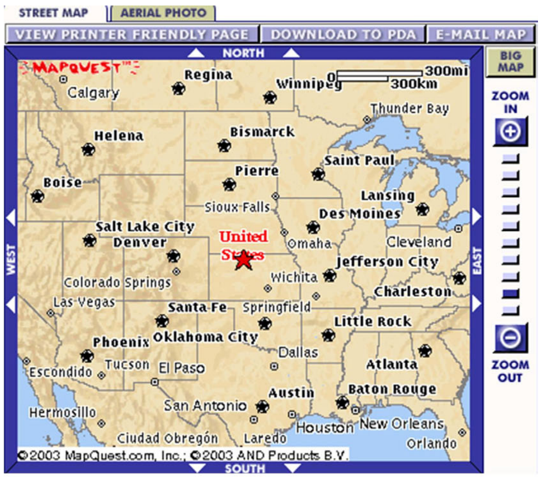
INTEL CORP (NasdaqNM:INTC) Quote data by Reuters Edit

After Hours (RTM/ECN): 29.57 ↓0.10 (0.34%)

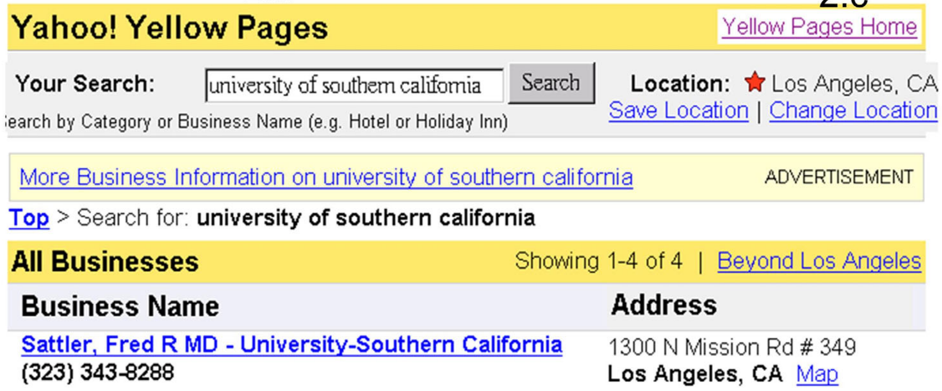
Last Trade:	29.67	Day's Range:	29.50 - 30.00
Trade Time:	4:00PM ET	52wk Range:	12.95 - 29.99
Change:	↓0.28 (0.93%)	Volume:	41,163,928
Prev Close:	29.95	Avg Vol (3m):	57,184,181
Open:	29.92	Market Cap:	193.15B
Bid:	17.88 x 100	P/E (ttm):	56.51
Ask:	41.47 x 100	EPS (ttm):	0.53
1y Target Est:	31.56	Div & Yield:	0.08 (0.27%)

Get streaming real-time charts in MarketTracker.

2.a



2.c



Yahoo! Yellow Pages | Yellow Pages Home

Your Search: university of southern california Search | Location: Los Angeles, CA
Search by Category or Business Name (e.g. Hotel or Holiday Inn) | Save Location | Change Location

More Business Information on university of southern california ADVERTISEMENT

Top > Search for: university of southern california

All Businesses Showing 1-4 of 4 | Beyond Los Angeles

Business Name	Address
Sattler, Fred R MD - University-Southern California (323) 343-8288	1300 N Mission Rd # 349 Los Angeles, CA Map

2.b

Figure 2: Some limitations of the online services: 2.a shows that the ticker symbol is required for Yahoo Finance. 2.b indicates that Yahoo Yellow page does not return the ZIP code. 2.c points out that Mapquest cannot produce the correct map of the Los Angeles city while the typo inputs "Los Angels" is used.

[4] http://www.mapquest.com/

3. The SERGEANT framework for handling the limitations of online services

In this section we describe SERGEANT, a framework to develop flexible web agents that are adaptive to the limitations of the online services described in the previous section. The key idea lies in exploiting a search engine as an auxiliary tool that generates required information.

3.1 The assumption

Our approach is appropriate for agents utilizing online services that have inputs and outputs satisfying the following assumption: *Given $E=\{e_1...e_n\}$ is an input or output set of an online service, then $\forall e_i \in E, \exists$ a non-empty set E' as a subset of E, and $E' \neq \{e_i\}$ s.t. all the elements in the set $E' \cup e_i$ appear somewhere in at least one document that can be found by a search engine.*

This assumption captures the idea that the elements in the input set are correlated with each other in the sense that we can use some of them to index the others through a search engine. The same assumption applies to the output set as well. The inputs and outputs of a typical online service often satisfy this assumption. For instance, the {title, director, cast} as the inputs to a movie site; the {street number, street name, zip code} as the outputs to a theater or restaurant lookup services as well as the inputs to a map lookup page, and the {title, author, publisher} as the inputs to the electronic library.

This assumption is similar to the fundamental assumption behind all information retrieval systems. It is a reasonable assumption in view of the fact that the inputs themselves are used together to query a set of outputs. So these inputs are to some extent correlated with each other through the outputs, and in many cases the correlations are even stronger.

However, we do not assume similar correlation to occur between inputs and outputs, which would be a much stronger assumption than the one we made and conceivably can be satisfied in fewer cases. In other words the assumption does not necessarily hold if the set E is the union of input and output sets. Take a geocoder for example: the inputs (address) and outputs (latitude and longitude) usually do not appear together in any documentation that can be found by a search engine.

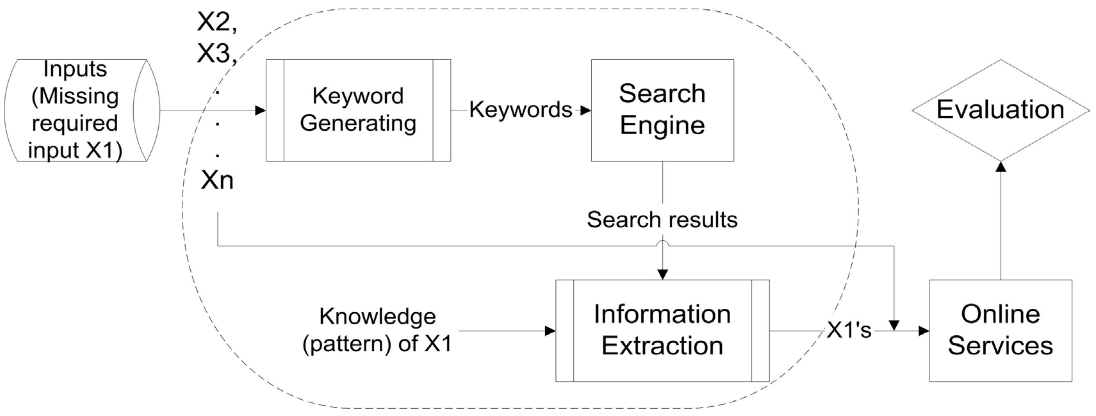


Figure 3. SERGEANT framework to handle missing but required input X1

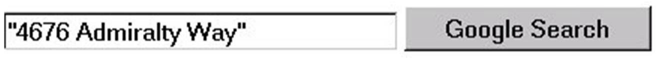
3.2 Handling required input and incomplete output limitations

To deal with the first two limitations of an online resource, we propose the SERGEANT framework (the components circled in Figure 3), which embraces the idea of utilizing the search engine along with the known information to generate potential candidates for the missing input and output fields.

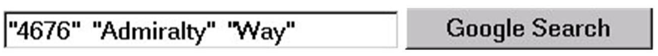
Figure 3 displays how the SERGEANT framework can be used as a pre-processor to produce missing but required input fields. SERGEANT works in three stages to generate the required inputs. The first stage is keyword-generation. In this stage, the agent uses incomplete input data to form a set of keywords to the search engine: Given an incomplete set of input fields $x_2 \ldots x_n$ (x_1 is the missing but required input field), the web agent can formulate the strictest keywords by concatenating them (keyword=”$x_2 \ldots x_n$”). Alternatively it is feasible to relax the keyword by quoting each before combining them (keyword=”x_2”,…,”x_n”). It is also feasible to further reduce strictness by dropping some inputs (e.g. keyword=”x_2”,…,”x_i”, $i<n$; or keyword= “x_2”, “x_4”…” x_n”). Figure 4 gives an example showing how the keywords are generated given two known input field: “street number”=4676 and “street name”=”Admiralty Way”. Additionally one can apply the “keyword spices” [5] approach to perform domain specific searches through a general purposed search engine by adding some auxiliary keywords in this stage.

The second stage of SERGEANT is to query the search engine with one of the generated keywords for relevant documents.

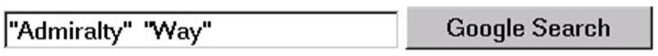
The third stage is to extract potential candidates for missing inputs from the documents returned by the search engine. After the candidates are generated, the agent can apply them to query the online services. Note that multiple input candidates could be generated; therefore the online services could return a set of plausible results instead of one. Producing more than one candidate results is reasonable since the user might prefer having multiple choices given some key criteria are missing.



4.a:Concatenate all the input fields



4.b:Combine each individual input fields seperately



4.c Use only a subset of input fields

Figure 4 Examples of different keyword generating strategies

Let us look at an example. Consider a movie service that takes “director name”, “leading actor”, and “leading actress” as inputs and outputs the movie title. If our agent is given only partial inputs “leading actor=P1, leading actress=P2” but not the required "director name", it will formulate different keywords ("P1 P2", "P1" "P2", "P1", or "P2") to the search engine, and then extract all the potential director names Dk in the third stage. The resultant sets (P1,P2,D1),…,(P1,P2,Dk) can then be applied to query the movie service one by one thus the user could be shown all the associated directors and movie titles returned based on P1 and P2.

The same concept can be applied to find the missing outputs by utilizing SERGEANT as the post-processor to the online service (Figure 5).

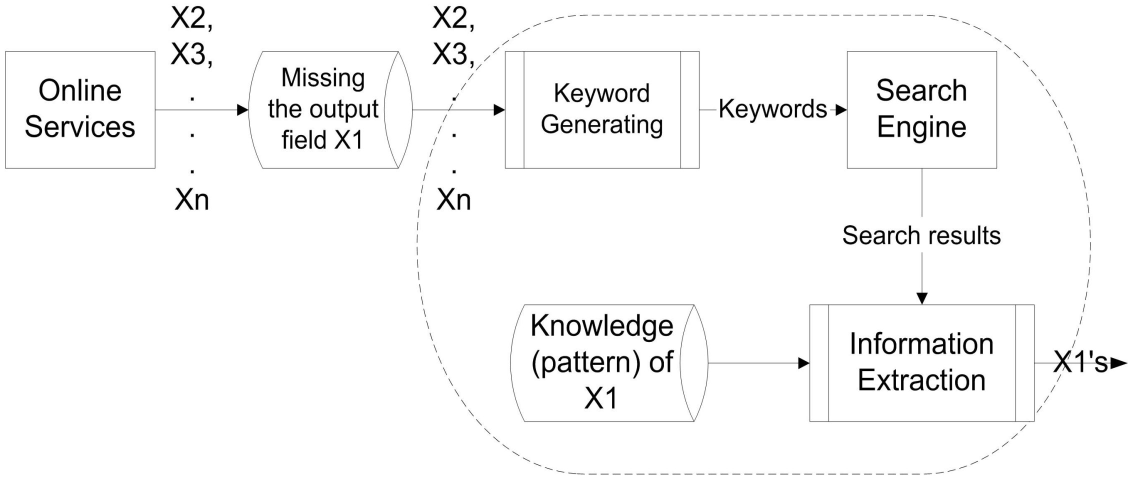


Figure 5 Using SERGEANT to fulfill the incomplete output fields

In the last stage of SERGEANT, we need to perform an Information Extraction (IE) task. This is the most challenging step in our framework with the goal to extract relevant facts from a document [6]. One traditional method for IE is to apply natural language processing techniques, which have been studied for decades [7, 8]. Alternatively, we can use wrapper technology to automatically wrap structured or semi-structured pages [9, 4]. In general supervised learning techniques are applicable for both semi-structured or non-structured sources [10, 11].

Extracting precise information from arbitrary web documents is generally a challenging problem. Our system, however, needs only the required inputs or incomplete outputs, which themselves usually form some pattern together with the given data (e.g. there are patterns for addresses or emails), which can be handled by techniques similar to named-entity tagging problems. Methods utilizing Hidden Markov models [12], rule-based systems [13], or Maximum Entropy models [14] to extract names, time, and monetary amounts are applicable approaches for our IE stage. The IE stage is not as difficult as a typical IE problem because the precision of our results is not critical. The backend online service can be treated as a precise evaluation engine that filters out the imprecise or irrelevant input sets generated by the IE engine.

Due to the facts that the pattern is known and the precision is not as important as recall in our IE stage, we present a suitable IE method as formatting the pattern instantiation problem into an AI Constraint Satisfaction Problems (CSP) [15] by modeling the pattern as a set of constraints. The advantage of formatting an IE problem into a CSP is that the recall and precision can be easily controlled by manipulating the constraints. Strict constraints imply high precision (and low recall) while sparse and loose constraints raise the recall at the cost of precision. For a recall-driven problem like the one we face, it is not necessary to explicitly program how to extract each individual field precisely in the pattern. Instead we tell the CSP engine what the pattern roughly looks like and the CSP engine will look for all the matched instances for us. This approach simplified the implementation of our recall-driven IE stage. As will be shown in section 4.2, with a backend online service as a verification component, we can find the missing address fields without having to know exactly how to extract the street name from a document.

3.3 Handling inverse queries

The third limitation of online services is that most of them accept only one-way queries. In this section we propose the idea of integrating SERGEANT into the forward service to construct a web agent that resolves inverse queries.

The challenge of constructing a reversible service lies in the fact that the original resource (online service) is an **unknown one-way function**. We first give a working definition, borrowed from cryptography, to a one-way function [16]:

Definition: *A function f from a vector space X to a vector space Y is called a one-way function if f(x) is "easy" to compute for all vectors $x \in X$, but for a random vector $y \in f(x)$ it is computationally infeasible to find any $x \in X$ satisfies f(x) = y*

In general there is no shortcut to find out the x that satisfies f(x) = y given y if the f(x) is a black box. The only way is to try the candidates inside X one by one until a match is found. This is also the basic assumption behind information security and key encryption/decryption [16]: the non-deterministic inverse function plus an immense input domain limit the chance of successful cracking (find x that satisfies f(x) = y) to almost zero.

Not knowing what is inside the black box of an online service, our strategy to improve the performance of inverse mapping is to ensure that the plausible candidates can be tested earlier. In this scenario the search engine plays a role as a heuristic generator, which provides the most plausible input candidates (Figure **6**).

Originally, the "trial and error" method has the input cardinality as large as the cardinality of the cross product of all input fields $|x_1|*|x_2|*\ldots*|x_n|$, where $|x_i|$ stands for the cardinality of a certain input field. There are two steps we take to trim the search domain in our framework:

1. Check if there exist related online services mapping the output y to some individual input field. If there is a service that takes y (or a subset of y) as inputs and generates a subset of x, say x_1 to x_k ($k<n$), then we need only to try as many as $|x_{k+1}|*|x_{k+2}|*\ldots*|x_n|$ different candidates.
2. Utilize the identified input fields $x_{1\ldots}x_k$ to indicate the remaining input fields $x_{k+1\ldots}x_n$ in SERGEANT.

As an example, assume there is a one-way online movie service that enables the users to search for a movie title by its leading actor, actress and director. To perform the inverse query, the very naive way is to test all the combination of actors, actresses and directors in the world and check which combination generates the given movie title. This naive method has the trial-and-error domain as large as |Director|*|Actor|*|Actress|. However, in step one we can first check if there are online services that map the movie title to some of the individual inputs (director, actor, or actress). Assuming we have found a service that maps the movie title to its director, then the cardinality of search space can be reduced to |Actor|*|Actress| after the first step is performed.

According to our fundamental assumption that the inputs are more or less correlated, heuristically in the "trial and error" period we would like to assign a higher priority to the input set that has elements associated with one another. In our second step, SERGEANT plays a role as this heuristic engine in the following manner: First use the identified director name as a keyword to the search engine. Then extract all the probable (actor, actress) pairs from each document returned by the search engine. Finally a set of plausible inputs fields of cardinality |(Actor, Actress) pairs given Director| will be generated. The |X given Y| represents the cardinality of X returned by the search engine given Y is used as the keyword. Conceivably, the number of "actor and actress" pairs associated with a director is much smaller than the total number of actors times the total number of actresses in the world. In this scenario SERGEANT acts as a heuristic function to guide the "trail and error" testing.

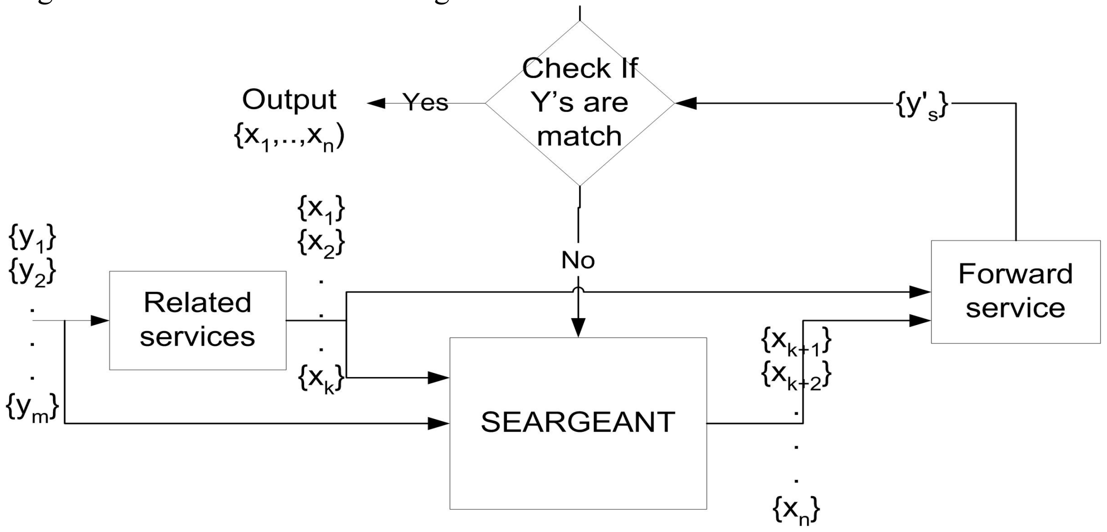


Figure 6 Exploiting SERGEANT to handle reverse query

3.4 Handling incorrect inputs

While utilizing an online service, the users or web agents occasionally encounter the situation that no output is generated due to the fact that the provided inputs contain errors (e.g. a typo). Nevertheless, most of the users would rather see something more or less related to their expected results even if there are errors in the inputs. In this section we propose a fault-tolerant web agent built on top of SERGEANT that handles minor error in the inputs.

The key concept is simple: If no output is generated given a set of the inputs, the system will assume one of the input fields is wrong. Consequently it can pick one input field to treat as unknown, use SERGEANT to generate the candidate values of this field (given this omitted field is a required input field), and then execute the online services by the new input sets. If the omitted input field truly contains error, then chance would be great that one of the new input candidates generated by SERGEANT will produce meaningful results. In this problem the SERGEANT framework is utilized as a data-cleaning tool while the online services again act as the evaluation engine (Figure 7).

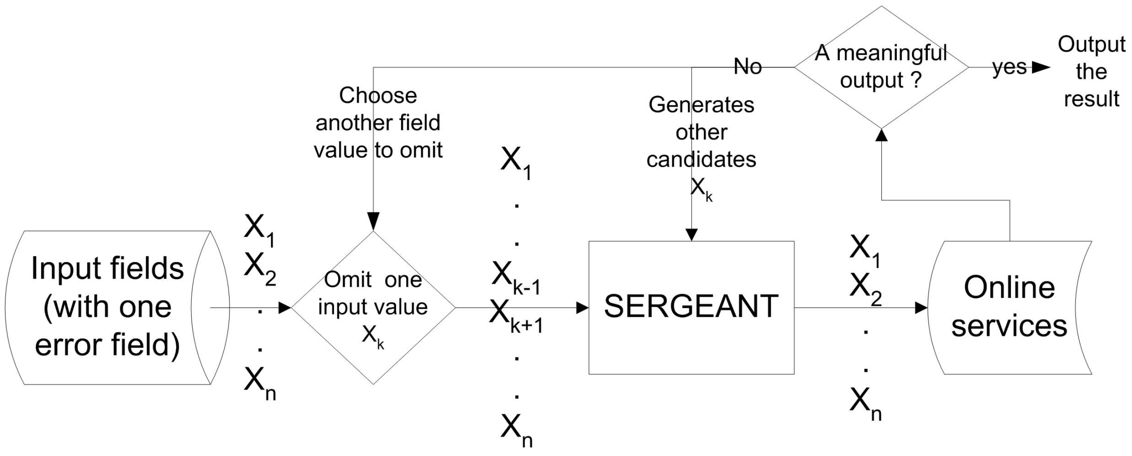


Figure 7 Integrating a search engine to handle incorrect inputs

4. Case Studies

We have implemented two web agents, the inverse geocoder and the address lookup module, as the demonstration for SERGEANT.

4.1 The inverse geocoder

The inverse geocoder is a web agent realizing the idea of developing inverse service by its forward source. We developed it by integrating SERGEANT with the online resource (Mapblast[5]) to transform the geocode into its equivalent address including the closest street number.

The inverse geocoder consists of three parts as shown in Figure 8: The city/state locator, the street name locator, and the street number locator, as a typical address in the United States can be uniquely identified by these three types of information.

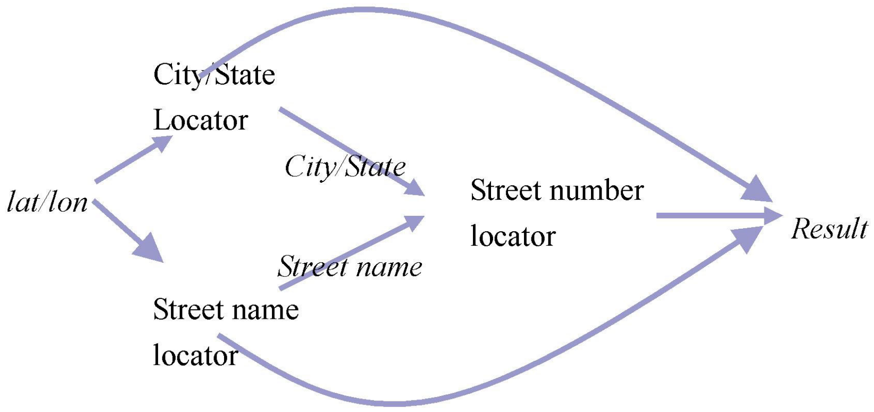


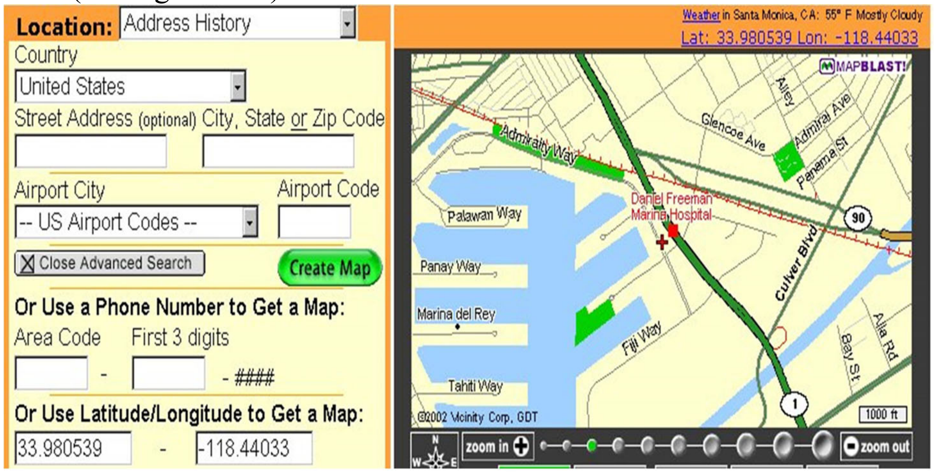
Figure 8 Data flow in the inverse geocoder

City/State locator: The corresponding zip code of a given geocode can be found using Mapblast site. Mapblast_Maps has the feature of displaying the map centered at a geocode given by its user. Checking the source code of this map page, we can find a hidden field “zip” that contains the zip code (Figure 9.a, 9.b). After using the lat/long pair

[5] http://www.mapblast.com

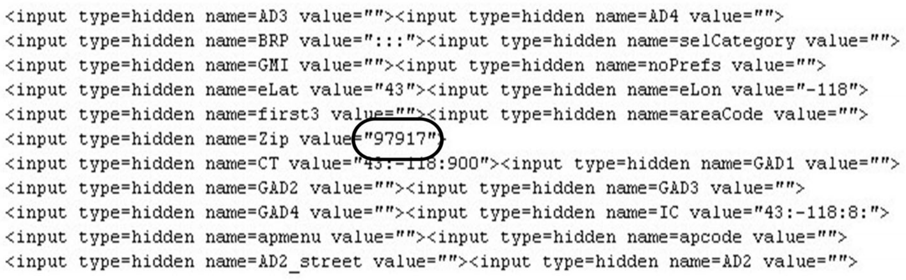
to query this Map service and wrapping the zip code, our system sends the zip code to USPS (United States Postal Service) site for corresponding city and state (Figure 9.c).

Street name locator: The street name finder discovers the street name of a given geocode by manipulating the inputs to the Mapblast_Direction. Mapblast_Direction is a service that returns the driving direction (in both text and graph format) from a user-specified starting point to a specified ending point. In its advanced search it allows users to use latitude and longitude to identify the points. The street name finder uses the original latitude and longitude as the starting point to the Mapblast_Direction. For the ending point, it uses the same latitude but slightly modifies the longitude to longitude minus 0.001 (see Figure 10.a).

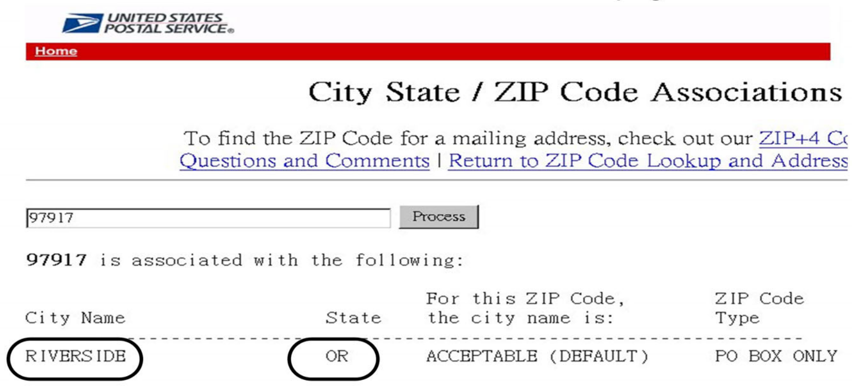

Location: Address History
Country: United States
Street Address (optional) City, State or Zip Code
Airport City: -- US Airport Codes -- Airport Code
Close Advanced Search
Create Map
Or Use a Phone Number to Get a Map:
Area Code First 3 digits - ####
Or Use Latitude/Longitude to Get a Map:
33.980539 - -118.44033

Weather in Santa Monica, CA: 55° F Mostly Cloudy
Lat: 33.980539 Lon: -118.44033

9.a The map feature of Mapblast


```
<input type=hidden name=AD3 value=""><input type=hidden name=AD4 value="">
<input type=hidden name=BRP value=":::"><input type=hidden name=selCategory value="">
<input type=hidden name=GMI value=""><input type=hidden name=noPrefs value="">
<input type=hidden name=eLat value="43"><input type=hidden name=eLon value="-118">
<input type=hidden name=first3 value=""><input type=hidden name=areaCode value="">
<input type=hidden name=Zip value="97917">
<input type=hidden name=CT value="43:-118:900"><input type=hidden name=GAD1 value="">
<input type=hidden name=GAD2 value=""><input type=hidden name=GAD3 value="">
<input type=hidden name=GAD4 value=""><input type=hidden name=IC value="43:-118:8:">
<input type=hidden name=apmenu value=""><input type=hidden name=apcode value="">
<input type=hidden name=AD2_street value=""><input type=hidden name=AD2 value="">
```

9.b The html code of the above page


UNITED STATES POSTAL SERVICE®

Home

City State / ZIP Code Associations

To find the ZIP Code for a mailing address, check out our ZIP+4 Co
Questions and Comments | Return to ZIP Code Lookup and Address

97917 Process

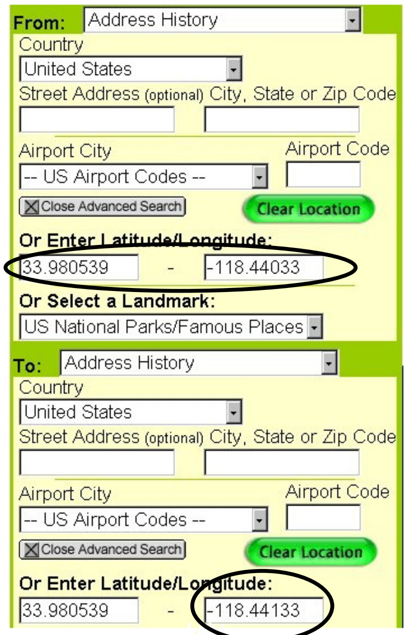
```
97917 is associated with the following:

                                      For this ZIP Code,        ZIP Code
City Name                   State     the city name is:         Type
--------------------------------------------------------------------------
RIVERSIDE                   OR        ACCEPTABLE (DEFAULT)      PO BOX ONLY
```

9.c Mapping the zip code to city and state through USPS site

Figure 9 How the city/state locator works

By slightly modifying the geocode as the destination point, it essentially asks the system to produce the driving direction from the original lat/long to a place that is closely adjacent to it. Usually, the street name returned in the driving direction is the street name of that lat/long (see Figure 10.b). The system also extracts the street direction since it is useful in the next step.

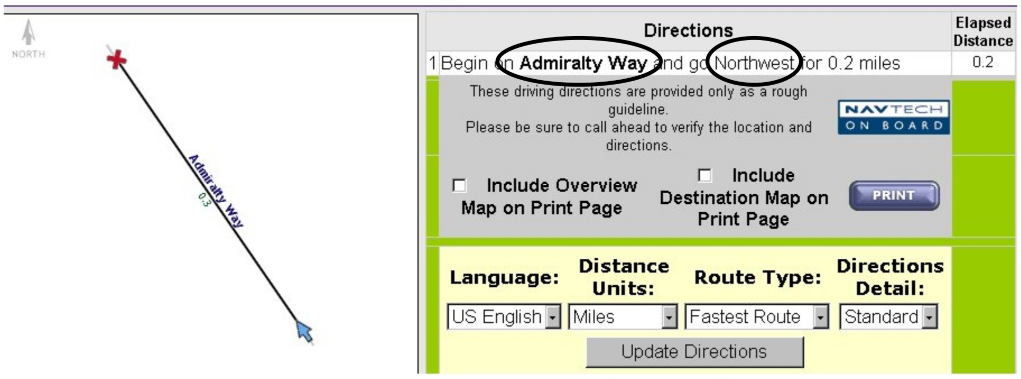
The city/state locator and street name locator realizes the idea proposed in step 1 of section 3.3 to use related services to acquire partial inputs from the outputs. Though there is no online service producing exactly what we want, both locators still manage to find out the information indirectly.



From: Address History
Country: United States
Street Address (optional) City, State or Zip Code
Airport City: -- US Airport Codes -- Airport Code
Close Advanced Search | Clear Location
Or Enter Latitude/Longitude: 33.980539 - -118.44033
Or Select a Landmark: US National Parks/Famous Places

To: Address History
Country: United States
Street Address (optional) City, State or Zip Code
Airport City: -- US Airport Codes -- Airport Code
Close Advanced Search | Clear Location
Or Enter Latitude/Longitude: 33.980539 - -118.44133

(a)



	Directions	Elapsed Distance
1	Begin on Admiralty Way and go Northwest for 0.2 miles	0.2

These driving directions are provided only as a rough guideline.
Please be sure to call ahead to verify the location and directions.

☐ Include Overview Map on Print Page
☐ Include Destination Map on Print Page
PRINT

Language: US English | Distance Units: Miles | Route Type: Fastest Route | Directions Detail: Standard
Update Directions

(b)

Figure 10: (a) demonstrates the inputs for street name locator. Note that the destination is very close to the starting point. (b) shows the output of Mapblast, the street name returned is the street name belongs to that latitude/longitude

Street number locator: As shown in Figure 8, the street number locator takes the street name and city/state as inputs and outputs the street number of the corresponding lat/long. The idea lies in that if we can somehow find two valid street numbers with respect to the given street name and city/state, then it is possible to utilize a forward geocoder to geocode them as two reference points and apply interpolation (given the address is in between the two reference points) or extrapolation (when the address is not in between the two reference points) method to pinpoint the exact street number. For example, to locate the street number of the geocode (33.9803, -118.4402) given the known street name "Admiralty Way" and zip code "90292", we can first geocode two reference addresses of that street, say "4000 Admiralty Way, 90292" (33.9815, -118.4599) and "5000 Admiralty Way, 90292" (33.9791, -118.4522). The interpolation method can then be applied on latitude or longitude[6] to find the target street number as 4511. Figure 11 indicates the equation of interpolation. If the street number is not uniformly distributed, then it is necessary to repeat the same procedure iteratively until it converges.

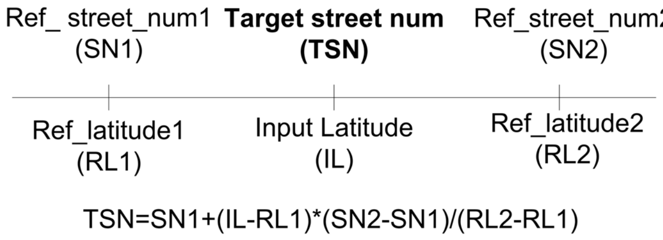


Figure 11. Interpolation on latitude

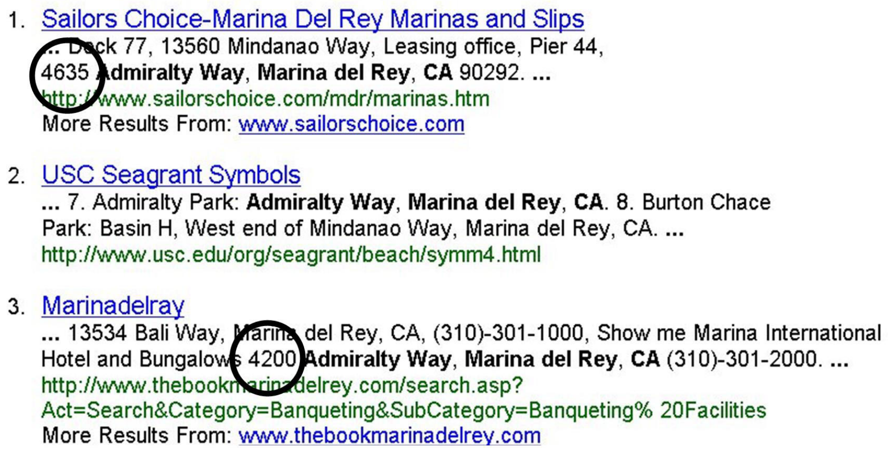
The tricky part of this approach lies in how to find the first two valid reference points. Randomly picking street numbers is not efficient due to the variety of the street numbers. For example in United States some streets have valid street numbers from 1 to 100 (e.g. Mason St, Coventry, CA) while others have valid numbers only between 34000 to 38000 (e.g. Ridge Rd, Willougby, OH). To resolve this problem we applied SERGEANT as proposed in section 3.3 to reduce the cardinality of the trial-and-error domain. The idea arises from the fact that the street numbers indexed by the street name and zip code through the search engine have high possibility to be valid street numbers for that street name and zip. Figure 12 shows the results returned by the Yahoo[7] search engine while using a street name "Admiralty Way" and zip code "90292" as the keyword. The street number locator extracts the street numbers returned by Yahoo (4635 and 4200 in this case) as the reference points.

The street number locator brings SERGEANT into play to prune the size of "trial and error" domain of the street number given the street name and zip code. It realizes the idea of the second step in section 3.3 by applying the search engine as a heuristic function to guide the testing procedure.

Evaluation: The evaluation is designed in the following manner: 50 restaurant and 50 residential addresses from 10 different cities in United States are used as the test addresses. We first determine the lat/long of these test addresses. These geocodes can then be used as the inputs to our reverse geocoder and the results will be verified by

[6] Whether using latitude or longitude for interpolation depends on the orientation of the street: the latitude is used for a north-south street, and the longitude is applied elsewhere.
[7] http://www.yahoo.com

comparing them with the original addresses. 90 out of the 100 geocodes are transformed back to the exact address to the level of street number. The reason for the incorrect transformations comes from the inconsistency between the forward geocoder and driving direction in Mapblast. In other words the street name returned by Mapblast_Driving is not the actual one.



1. Sailors Choice-Marina Del Rey Marinas and Slips
... Dock 77, 13560 Mindanao Way, Leasing office, Pier 44,
4635 **Admiralty Way**, **Marina del Rey**, **CA** 90292. ...
http://www.sailorschoice.com/mdr/marinas.htm
More Results From: www.sailorschoice.com

2. USC Seagrant Symbols
... 7. Admiralty Park: **Admiralty Way**, **Marina del Rey**, **CA**. 8. Burton Chace Park: Basin H, West end of Mindanao Way, Marina del Rey, CA. ...
http://www.usc.edu/org/seagrant/beach/symm4.html

3. Marinadelray
... 13534 Bali Way, Marina del Rey, CA, (310)-301-1000, Show me Marina International Hotel and Bungalows 4200 **Admiralty Way**, **Marina del Rey**, **CA** (310)-301-2000. ...
http://www.thebookmarinadelrey.com/search.asp?
Act=Search&Category=Banqueting&SubCategory=Banqueting% 20Facilities
More Results From: www.thebookmarinadelrey.com

Figure 12. Extracting the street number by the street name and city/state

The efficiency of our inverse geocoder depends on the network conditions. Therefore, instead of using the elapsed time, we calculated how many times on average each service is called before the addresses are generated. The results are listed in Table 2. For most of our test cases the search engine does find two valid street numbers, which enables the system to perform the forward geocoder as few as three times (two for geocoding the reference points and one for verifying the inter/extrapolated result). There are 3 cases that SERGEANT could not find two valid street numbers. In those cases the system had to randomly pick the street numbers and verify it by the forward geocoder. We had to execute the forward geocoder on an average of 10 times in order to get the valid address in these 3 cases, compared with an average of 4.5 times for the rest. This fact signifies that using SERGEANT can reduce the number of times to query the forward geocoder.

Table 2 The average number of times each service is performed while performing an inverse query

online services	Number of Times called
Mapblast:Map	1
Mapblast: Direction	1
Yahoo Search	1
Mapblast:forward geocoder	4.7

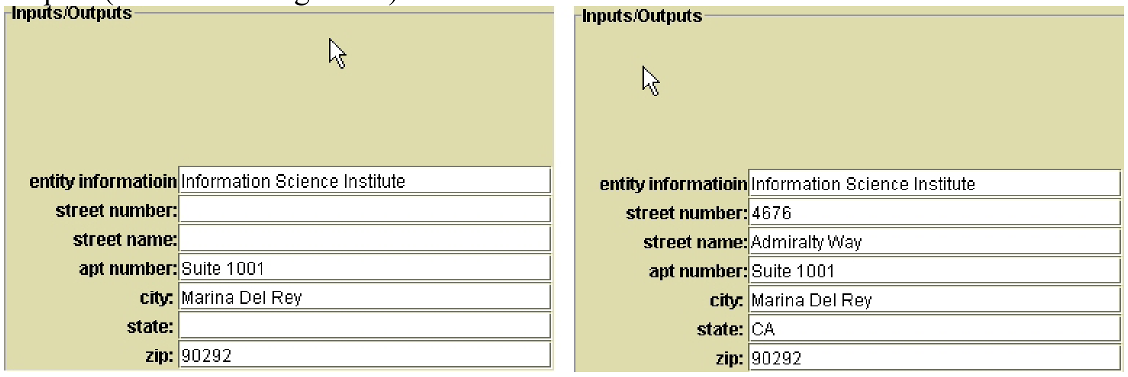
Our inverse geocoder does not require a local database nor intensive computation. Moreover, it can be implemented in a short time. The zip finder and street name finder demonstrate the fact that there is considerable information hidden on the Web, only one has to create ways to find it. The street number locator is a demonstration of applying SERGEANT as a heuristic function to improve the efficiency.

4.2 Address lookup module

In this case study we demonstrate how SERGEANT can be applied to handle the limitations related to inputs and outputs. The test platforms used are the WhitePages[8] service and the Yahoo Yellowpages[9]. The WhitePages site requires complete addresses to obtain the corresponding phone numbers. Yahoo Yellowpages is a service that generates address information as outputs, but the zip code information is missing in its output.

We implemented an address lookup module as the preprocessor to the WhitePages to fulfill all the required input fields given only partial address information. The same module is used as a post-processor to YellowPages for the missing zip codes.

This module has seven optional input fields: the entity information, street number, street name, apartment number, city, state, and zip code. It has seven identical fields as outputs (as shown in Figure 13).



entity informatioin	Information Science Institute
street number:	
street name:	
apt number:	Suite 1001
city:	Marina Del Rey
state:	
zip:	90292

entity informatioin	Information Science Institute
street number:	4676
street name:	Admiralty Way
apt number:	Suite 1001
city:	Marina Del Rey
state:	CA
zip:	90292

Figure 13 Sample inputs (left) and outputs (right) for address lookup module

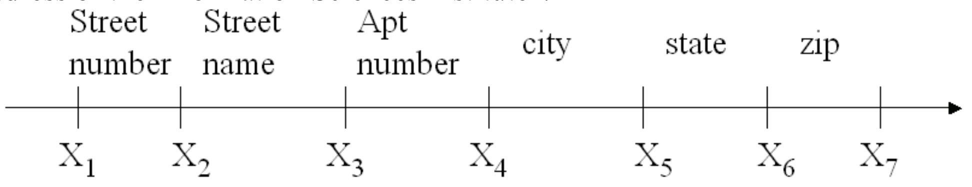
The idea is to utilize a search engine with the known fields to identify the missing fields. Note that in addition to the address, the user can provide any other necessary information in "entity information" field and this auxiliary information will be treated as the keyword to the search engine as well.

The keyword-generating stage takes the inputs to generate the keywords from the strictest one to the loosest one, as described in section 3.2. In the search phase the module uses the keyword to retrieve relevant pages. It then sends the top 100 ranked documents returned by the search engine to the third phase. The third IE phase is designed to extract the potential candidates of address from these documents. The addresses of the United States form a regular pattern: a street number followed by a street name followed by an apartment number, then the city, state, and zip.

Given this pattern and some intrinsic characteristics of each address field, we can format this address extraction problem into an equivalent constraint satisfaction problem (CSP). Figure 14 provides an example how an information extraction problem can be represented as a CSP problem. In the corresponding CSP problem, the starting positions of each field ($x_1 \ldots x_7$) in a document are variables. The inputs to the CSP engine are the constraints (which encode the knowledge about the U.S. address as shown in Figure 14.a) and a document returned by the search engine (assuming it begins with "The address of

[8] http://www.whitepages.com/address-lookup
[9] http://yp.yahoo.com/

the information sciences Institute is 4676 Admiralty Way, #1002, Marian Del Rey, CA, 90292" as shown in 14.b). The CSP engine will automatically generates all feasible bindings on the variables, e.g. $(X_1,X_2,X_3,X_4,X_5,X_6,X_7)$=(9,10,12,13,16,17,18) or $(X_1 \ldots X_7)$=(1,2,3,4,5,6,7). For example, X_4=13 and X_5=16 reveals that the "city" is in between the 13th and 16th word in the document (which is "Marina Del Rey" in this example). Therefore the result $(X_1 \ldots X_7)$= (9,10,12,13,16,17,18) can be mapped to the address "4676, Admiralty Way, #1001, Marina Del Rey, CA, 90292" as shown in Figure 14.c, while the result $(X_1 \ldots X_7)$= (1,2,3,4,5,6,7) will be mapped to an inferior output "the address of the Information Sciences Institute".



14.a

The address of the Information Sciences Institute is 4676 Admiralty Way, #1001, Marina Del Rey, CA, 90292

14.b

X's	X_1	X_2	X_3	X_4	X_5	X_6	X_7
Binding of X's	9	10	12	13	16	17	18

Address fields	Street number	Street name	Apt number	city	state	zip
Extracted results	4676	Admiralty Way	#1001	Marina Del Rey	CA	90292

14.c

Figure 14: Mapping an IE problem to a CSP problem

The constraints in CSP come from the order shown in the pattern (e.g. in general the city name is followed by Apt#, thus x4-x3>=0) as well as the characteristic of each input type (e.g. zip codes are numbers of 5 digits, or state names have at most two words). The constraints can be divided into two categories: the precise constraints and the flexible constraints (Table 3). The precise constraints represent the certain information. For example, "$x_n \leqq x_{n+1}$" represents the fact that the order of the fields of U.S. address cannot be altered while some of the fields can be absent; "W(x1,x2) is a number" represents the fact that a street number can only be a number. The flexible constraints represent the uncertain information that can be adjusted to control the recall and precision. For example, x3-x2 $\leqq$k1 controls the length of a street name and "W(x2,x3) is of type K" constraints the content of the apartment number. Assigning larger value to k1 will improve the recall at the cost of precision. Similarly “K can be only a number” has lower recall than “K can be a number or a number with a character (such as B23)”.

By representing this knowledge as a set of constraints, our CSP engine can generate all the consistent variable sets of (x1…x7) satisfying these constraints. Each solution indicates a position of a potential address pattern in the document. In our system determining the precise constraints are not necessary. The major goal of our IE stage is to improve the recall. The reason is that we have a backend online service as a precise verification tool. The CSP approach is suitable for the IE stage in SERGEANT since it provides a simple way to control the recall and precision by manipulating the constraints.

Table 3: Two different sets of constraints.

Precise constraints	Flexible constraints
$x_n \le x_{n+1}$	x3-x2 ≤k1
x4 - x3 ≤1	1≤x5-x4≤k2
W(x1,x2)[10] is a number	W(x3,x4) is of type K

The addresses generated by the CSP engine will then be sent to the evaluation engine for ranking. They are ranked according to their similarity to the known input fields. The agent can then use the ranked outputs to query online services such as the WhitePages and the Yahoo Yellowpages.

The evaluation procedure is as shown in Figure 15:

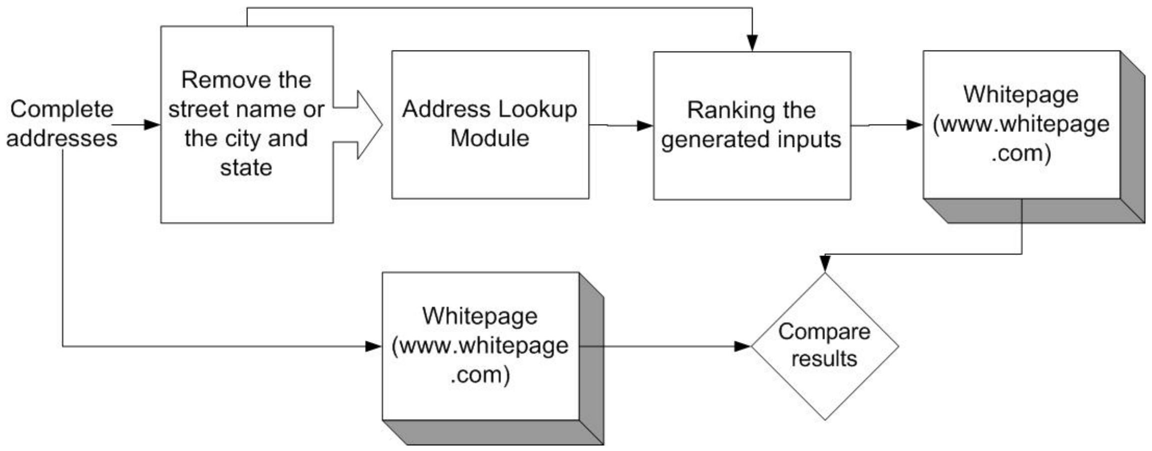

Complete addresses
Remove the street name or the city and state
Address Lookup Module
Ranking the generated inputs
Whitepage (www.whitepage .com)
Whitepage (www.whitepage .com)
Compare results

Figure 15 The experiment designed to evaluate the address lookup module

First we utilize a set of valid addresses to query the Whitepage site for the corresponding phone numbers. Then we manually remove some of the input fields and apply these incomplete addresses as the inputs to our address lookup module. The address lookup module must discover the missing fields and sent the entire recovered address to query the Whitepage site for the phone number. We have a ranking program that ranks the outputs of address lookup module simply by checking the similarity of them to the inputs of the address lookup module. The module is evaluated by examining whether the retuned phone number generated by one of the top 3 ranked outputs match

[10] W(x,y) stands for the words between position x and y.

the true phone number. We have tested our module under two different scenarios for 50 valid addresses. Half of them missed the street name while the other half lack both the city and state information. The results show that in 70% of the cases our module can recover the missing street name (in other words, 70% of the correct street names appear in the top three ranked outputs). The task is simpler (90% accuracy) when the missing fields are city and state. The failure to achieve 100% accuracy is due to the fact that the missing information does not occur together with the known one in any online document (i.e. the assumption in section 3.1 was violated), thus there is no way for our program to discover it. We also apply our module as the post-processor to Yahoo Yellowpages. The results show that it can recover all the zip information with 100% accuracy.

In addition to the above functionality, we also implemented the error-detecting feature as described in section 3.4. That is when there is no meaningful output generated by the WhitePage, our program will assume that one input field contain error and apply SERGEANT to regenerate it. We tested it by manually generating one typo on the street or city name of a full address and verifying that whether the system can automatically detect and recover the typo. The results demonstrate that the typographical error can be detected by our system and 80% of them can be recovered. The failing 20% is again because the assumption of section 3.1 is not satisfied.

This case study indicates that our framework is applicable in designing flexible web agents that handles the required-input, incomplete-output, and incorrect-input limitations.

5. Related Work

The idea of exploiting the functionality of search engines resembles Etzioni's information food chain [17], in which the search engines are located in the middle of the food chain and there are goal-oriented softbots (software robots) built on top of them. MetaCrawler [17] is a meta-search engine built on top of several search engines. Citeseer [18] is an autonomous web agent that utilizes search engines for retrieving and identifying publications. WebSuite [19] has its own search engine that can accept not only keywords but also the criteria of the connection between pages (e.g. finding a web page that comes from page P via link N). Also most of the question and answering (QA) systems utilize the search engine as well [1, 2]. However, they operate the search engine as the major tool for querying information and do not usually integrate it with the other resources. In our approach the online services are still the major tool for acquiring information while the search engine plays a supporting role in providing the extra information and reducing the input cardinality for an inverse service.

On the other hand many platforms have made efforts toward integrating various online services to achieve specific tasks, such as ShopBot [20], the Information Manifold [21], Ariadne [22, 23] and the Venus and Mars System [24]. These systems are aimed at resolving the different issues of integrating data from the web such as information resource selection and modeling, view integration from distributed sources, and handling the inconsistency among sources. However none of these systems are focused on generally fixing the limitations of the resources. The search engine does not play a major role in these platforms.

Section 3.4 demonstrates that SERGEANT can serve as a data cleaning tool. Data cleaning research aims at resolving the incorrectness, incompleteness, and inconsistency on data. Kimball breaks down the data cleaning into six steps [25]: elementizing, standardizing, verifying, matching, householding, and documenting. Our approach of handling input errors presented in section 3.4 is essentially the procedure of verifying the correctness of the data. Several data cleaning methods were proposed to deal with similar problems: N-gram sliding window targets at handling spelling errors [26], and Kimball [25] proposed the approaches for name/address cleaning by looking for the associations between data as well as comparing different sources. One common characteristic for these approaches is that either local databases (e.g. dictionary or city/state mapping) or multiple resources for the same data are required. Our approach, by which the search engine is applied to find out a clue about which fields might be wrong, differs in a way that no local database is required, nor do we need to find and check multiple resources for the same data.

It is also feasible to integrate other online resources instead of SERGEANT to resolve the required input or incomplete output limitation. However, the SERGEANT framework of integrating a search engine into a web agent is more suitable for this situation because it saves the time of finding appropriate sources. Additionally it is more flexible and less risky since there any many available search engines and they are usually more stable than the online services.

6. Conclusions

In this paper we proposed the SERGEANT framework, which has three stages: the keyword generation stage, search stage, and information extraction stage. By integrating SERGEANT with a web agent, it is feasible to overcome four general limitations: the required-input limitation, the incomplete-output limitation, the non-reversibility limitation and the lack of fault-tolerance. Within the SERGEANT framework, we also present the idea of resolving the information extraction problems by translating them into an equivalent CSP, which simplifies the implementation of a recall-driven IE task such as the one required in SERGEANT. We implemented two flexible web agents as the demonstration for SERGEANT: The inverse geocoder is the first web agent that accomplishes the inverse geocoding task without employing any local database. The address lookup module demonstrates a flexible and reusable component that can be plugged into a variety of web agents using addresses as inputs and outputs.

7. References

[1]C. Kwok, O. Etzioni and D. S. Weld, *Scaling question answering to the web.* ACM Transactions on Information Systems, 2001. **19**(3): p. 242 - 262.

[2]R. Srihari and W. Li., *Information extraction supported question answering.* Proceedings of the 8th Text Retrieval Conference, 1999: p. 185-196.

[3]K. Lerman, S. N. Minton and C. A. Knoblock, *Wrapper maintenance: A machine learning approach.* Journal of Artificial Intelligence Research, 2003. **18**: p. 149-181.

[4]N. Kushmerick, D. Weld and R. Doorenbos, *Wrapper induction for information extraction.* Proc. of 15th International Conference on Artificial Intelligence, IJCAI-97, 1997: p. 729-735.

[5]S. Oyama, T. Kokubo, T. Ishida, T. Yamada and Y. Kitamura, *Keyword Spices: A New Method for Building Domain-Specific Web Search Engines.* the 17th International Joint Conference on Artificial Intelligence, 2001: p. 1457-1466.

[6]M. T. Pazienza, *Information Extraction: A multidisciplinary Approach to an Emerging Information Technology*, in *volume 1299 of Lecture Notes in Computer Science, International Summer School, SCIE-97.* 1997:Frascati (Rome), Springer.

[7]R. Basili, M. Vindigni and F. M. Zanzotto. *Integrating Ontological and Linguistic Knowledge for Conceptual Information Extraction*. in *Wed Intelligence 2003*. 2003. Halifax, CA.

[8]Y. Wilks, *Information Extraction as a core language technology*. 1997, In M-T. Pazienza (ed.): Springer, Berlin.
[9]I. Musela, S. Minton and C.A. Knoblock, *Hierarchical wrapper induction for semistructured information sources.* Autonomous Agents and Multi-Agent Systems, 2001: p. 93- 114.
[10]D. Freitag. *Information extraction from html: Application of a general learning approach*. in *the Fifteenth Conference on Arti cial Intelligence AAAI-98*. 1998.
[11]S. Soderland, *Learning Information Extraction Rules for Semi-structured and Free Text.* Machine Learning, 1999. **34(1-3)**: p. 233-272.
[12]M. Bikel. *Nymble: a high-performance learning name-finder*. in *Fifth Conference on Applied Natural Language Processing*. 1997: Morgan Kaufmann Publishers.
[13]G. R Krupka and K. Hausman. *IsoQuest Inc: Description of the NetOwl "Fext Extraction System as used for MUC-7"*. in *Seventh Machine Understanding Conference*. 1998.
[14]A. Borthwick, J. Sterling, E. Agichtein and R. Grishman. *NYU: Description of the MENE Named Entity System as Used in MUC-7*. in *Proceedings of the Seventh Machine Understanding Conference (MUC-7)*. 1998.
[15]S. Russell and P. Norvig, *Section 2.5 Constraint Satisfaction Problems*, in *Artificial Intelligence: A Modern Approach*. 2002, Prentice-Hall.
[16]A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of applied Cryptography*. 1996: CRC Press.
[17]E. Selberg and O. Etzioni. *Multi-Service Search and Comparison Using the MetaCrawler*. in *Proceedings of the 4th international World Wide Web Confernece*. 1996.
[18]K. D. Bollacker, S. Lawrence and C. L. Giles. *CiteSeer: An Autonomous web Agent for Automatic Retrieval and Identification of Interesting Publications*. in *2nd International ACM Conference on Autonomous Agents*. 1998.
[19]C. Beeri, G. Elber and T. Milo. *WebSuite -- A Tool Suite For Harnessing Web Data*. in *Proceedings of the International Workshop on the Web and Databases*. 1998. Valencia, Spain.
[20]R. B. Doorenbos, O. Etzioni and D. Ordille. *Querying Heterogeneous Information Sources Using Source Descriptions*. in *Intl. Conference on Very Large Data Bases (VLDB)*. 1996.
[22]C. Knoblock, S. Minton, J. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada, *The ARIADNE Approach to Web-Based Information Integration.* International Journal of Cooperative Information Systems, 2000: p. 145--169.
[23]C. Kwok and D. Weld. *Planning to gather information*. in *14th National Conference on AI*. 1996.
[24]Y. Kitamura, *Web Information Integration Using Multiple Character Agents*, in *Life-Like Characters: Tools, Affective Functions, and Applications*, M. Ishizuka, Editor. 2003, Springer-Verlag. p. 295-316.
[25]R. Kimball, *Dealing with Dirty Data*, in *DBMS*. 1996.
[26]K. Kukich, *Techniques for Automatically Correcting Words in Text.* ACM Computing Surveys, 1992: p. 377-439.