A Single-Significant-Digit Calculus for Semi-Automated Guesstimation

A Single-Significant-Digit Calculus for Semi-Automated Guesstimation*

Jonathan A. Abourbih, Luke Blaney, Alan Bundy, and Fiona McNeill

School of Informatics, University of Edinburgh
jabourbih@acm.org, L.Blaney@sms.ed.ac.uk, A.Bundy@ed.ac.uk, f.j.mcneill@ed.ac.uk

Abstract. We describe a single-significant-digit calculus for estimating approximate solutions to guesstimation problems. The calculus is formalised as a collection of proof methods, which are combined into proof plans. These proof methods have been implemented as rewrite rules and successfully evaluated in an interactive system, GORT, which forms a customised proof plan for each problem and then executes the plan to obtain a solution.

1 Introduction

The benefits of the huge amount of information available via the internet will not be fully realised until we can automatically combine it in novel ways. Unfortunately, most of the interest in information retrieval focuses around the extraction of isolated facts. Interest in question answering, which *did* try to combine such facts via inference, has waned recently (but see §6).

This paper tries to return to that earlier vision. We are interested in solving problems by using inference to combine information from a variety of sources, including linked data, natural language, etc from the internet, and knowledge already known to the user. Our research programme is as follows:

1. Construct a proof plan [3] that can be used to identify the information required to solve the current problem and then solve it.
2. Retrieve this information and use it to construct an initial problem-specific, logic-based ontology from multiple sources. Note that the sources might contain ungrammatical, uncertain and dubious information, and must be parsed, disambiguated, checked, etc and expressed logically. The final logical representation might be more sophisticated than any explicitly present in the original sources.
3. Execute the proof plan in the customised ontology to provide the required solution.

* This paper is based on the MSc project of Jonathan A. Abourbih and the undergraduate project of Luke Blaney. Dr McNeill was funded by ONR project N000140910467. We would like to thank 3 anonymous IJCAR referees for their constructive comments, and Aparna Ghagre for information on the QUARK system.

4. Sanity check the answer and the information used in its construction, e.g., ensure that the customised ontology is consistent. If a problem is diagnosed, use ontology evolution techniques to repair it, extracting new information, if necessary, then re-execute the plan.

1.1 Guesstimation

Guesstimation is the task of finding a single-significant-digit estimate to a quantitative problem based on a combination of intuition, facts, and reasoning. The types of problems that are suited to guesstimation are those for which a precise answer cannot be known or easily found [13, 10]. An example guesstimation problem is:

How many golf balls would it take to circle the Earth at the equator?

This can be answered by finding the diameter of a typical golf ball and the circumference of the Earth and dividing the latter by the former. More examples can be found in Table 1 in §5. Guesstimation requires a combination of facts, planning, reasoning, and guesswork. We have implemented a semi-automated guesstimator in the system GORT (Guesstimation with Ontologies and Reasoning Techniques) using SWI-Prolog, which was chosen for its excellent linked-data toolkit.

Since common patterns of reasoning can be identified and formalised as proof plans, and the information from a variety of diverse sources must be combined in unexpected ways, guesstimation is an ideal *initial* vehicle for realising the programme outlined above. We emphasise "initial" because the work described in this paper does not yet realise all aspects of our programme. For instance, we have not yet tackled fault diagnosis or ontology repair, although these are the topic of other research projects in our research group [8, 4]. Nor have we used natural language processing techniques.

1.2 The SingSigDig Calculus

According to [13], the normal form for guesstimation answers is a number in single significant digit form, $d \times 10^i$, in SI units, where d is a digit from $1, \ldots, 9$ and i is an integer. Where quantities are not originally in this normal form, numeric values must be approximated to the form $d \times 10^i$ and non-SI units must be converted to SI[1].

We now define a calculus for reasoning about numbers approximated to a single significant digit, which we will call the SingSigDig Calculus. This calculus is expressed as a set of rewrite rules between first-order terms that evaluate to numbers in this normal form, i.e., function values are expressed only approximately. These rewrite rules are based on an equality that is modulo this approximation, which we represent as $s \rightsquigarrow t$.

[1] In practice, the current GORT implementation does not stick to strictly SI units.

Let $\mathbb{R}^\sim = \{d \times 10^i | d \in \{1, \ldots, 9\} \wedge i \in \mathbb{Z}\}$ be the domain of normal form numbers. Let $nf^\sim : \mathbb{R} \mapsto \mathbb{R}^\sim$ be the function that converts a real number into its nearest single significant digit approximation. Observe that $\mathbb{R}^\sim$ is the quotient space $\frac{\mathbb{R}}{nf^\sim}$.

Upper-case letters represent sets; lower-case letters represent objects, and the notation $\|\ldots\| : Set(\tau) \mapsto \mathbb{R}^\sim$ approximates the number of elements in a set.

Formulae in the SingSigDig Calculus are first-order expressions whose domain of discourse consists of numbers in $\mathbb{R}^\sim$ plus everyday objects and sets of such objects. Functions and predicates defined on this domain are abstracted from those defined on $\mathbb{R}$. Suppose, without loss of generality, that all real number arguments of function f (predicate p) are initial, i.e., that $f : \mathbb{R}^m \times \tau^n \mapsto \mathbb{R}$ ($p : \mathbb{R}^m \times \tau^n \mapsto \mathit{bool}$), where τ is the type of any non-$\mathbb{R}$ arguments of f, if any, so $n \geq 0$. For every such function f (predicate p), we define a corresponding $f^\sim : \mathbb{R}^{\sim m} \times \tau^n \mapsto \mathbb{R}$ ($p^\sim : \mathbb{R}^{\sim m} \times \tau^n \mapsto \mathit{bool}$). In particular, we define the equality predicate $=^\sim : \mathbb{R}^{\sim 2} \mapsto \mathit{bool}$. In general, $f^\sim$ ($p^\sim$) can be defined in terms of f (p) as follows:

$$f^\sim(nf^\sim(r_1), \ldots, nf^\sim(r_m), t_1, \ldots, t_n) ::= nf^\sim(nf^\sim(r_1), \ldots, nf^\sim(r_m), t_1, \ldots, t_n)$$
$$(p^\sim(nf^\sim(r_1), \ldots, nf^\sim(r_m), t_1, \ldots, t_n) ::= nf^\sim(nf^\sim(r_1), \ldots, nf^\sim(r_m), t_1, \ldots, t_n))$$

These definitions ensure that $f^\sim$ ($p^\sim$) is uniquely defined on its first m numeric arguments[2]. In order to ensure uniqueness for the next n non-numeric arguments, we need to make the following assumption.

Assumption 1 *Similarity Assumption:*
For all functions $f^\sim$ (predicates $p^\sim$) and sets S, to ensure that $f^\sim(\ldots, S, \ldots)$ ($p^\sim(\ldots, S, \ldots)$) is uniquely defined then we assume that:

$$\forall s_1, s_2 \in S.\ f^\sim(\ldots, s_1, \ldots) =^\sim f^\sim(\ldots, s_2, \ldots)$$
$$(\forall s_1, s_2 \in S.\ p^\sim(\ldots, s_1, \ldots) \iff p^\sim(\ldots, s_2, \ldots))$$

Note that $=^\sim$ is actually $=$, but over $\mathbb{R}^{\sim 2}$ rather than $\mathbb{R}^2$. This makes $=^\sim$ an appropriate basis for a rewriting calculus, since it inherits from $=$ the properties required of rewriting, namely transitivity, monotonicity and stability. $\rightsquigarrow$ is a directed version of $=^\sim$. Where the context makes clear that an approximate function is being used, we will usually drop the $\sim$ superscript.

We will frequently want to specify some typical element of a set. To formalise this we will use Hilbert's ϵ operator, [5]. We will designate ϵS to be a typical representative element of the set S.

We will use polymorphic functions which apply to both objects and sets of those objects. If S is a set, the semantics of $f(S)$ is $f(\epsilon S)$. An exception to this semantic rule is the function $\|S\|$ described above, which returns the approximate number of elements in the set S.

[2] We are indebted to an anonymous referee for suggesting this version of the definition of $f^\sim$ ($p^\sim$) to ensure this property.

1.3 Hypothesis

The hypothesis that we seek to evaluate in this project is:

> *Proof planning in the* SingSigDig *Calculus can be successfully used to solve guesstimation problems.*

Our evaluation consists of implementing this technology in the system GORT and applying it to a representative sample of guesstimation problems (see §5). 'Success' here is mainly measured by the proportion of test set guesstimation problems to which GORT returns an accurate result. Secondary characteristics of success are GORT's efficiency and its adaptability to the information available to it. In §6 GORT is compared favourably to related systems.

2 Guesstimation Proof Methods

Proof plans consist of a configuration of proof methods. We now describe the proof methods that typically apply to guesstimation problems. These methods have been derived by introspection and examination of the worked problems from [13]. To date it has proven possible to express each of them as a, sometimes conditional, rewrite rule based on the $\rightsquigarrow$ relation. It is unclear whether this will continue to be the case for future methods. The methods described below are not an exhaustive list of all techniques that apply to guesstimation problems, but they do form the basis of the methods currently implemented in GORT.

We have divided the methods into *primary* and *secondary*. Primary methods form the initial part of each guesstimation proof plan and are applied manually using the web interface (see §4.3). Primary methods also often require user input of parameters. Secondary methods are used to complete the proof plan and are applied automatically by GORT (see §4.2). One exception to this classification is the *user interaction* method, which can either be called manually as a primary method or automatically, as a secondary method on backtracking when all other methods have failed.

2.1 Primary Methods

The Total Size Method The *total size* method is applicable in cases where a guesstimation question requires the total of some physical quantity over a set. The general form of this question asks for the sum total of a quantity for all elements in a set. Thus, this type of question can be expressed as $\sum_{s \in S} f(s)$, which is rewritten as,

$$\sum_{s \in S} f(s) \rightsquigarrow f(S) \times \|S\|, \tag{1}$$

where S is a set of non-numeric objects of type τ and f is a function $f : \tau \mapsto \mathbb{R}^{\sim}$. An example might be, *What area would be required if all humans in the world were put in one place?*. Here, S is the set of all humans and $f(s)$ is the area occupied by s.

The Count Method The *count* method is applicable in cases where a guesstimation question requires a count, $\|Small\|$, of a set of small objects, $Small$, that would exactly fit a larger object, big. An example might be, *How many golf balls would it take to circle the Earth?*, which is also worked as an example in Section 3.

We assume that the sum of some measurement g over the elements of $Small$ is equal to a measurement f of big. f and g will typically be the volume, length, duration or mass of these objects. The units of the measurements must be the same. We can formalise the *count* method as:

$$g(Small) \neq \emptyset \wedge f(big) =^{\sim} \sum_{s \in Small} g(s) \implies \|Small\| \rightsquigarrow \frac{f(big)}{g(Small)}. \tag{2}$$

The preconditions of the *count* method are checked by user interaction; the user instantiates f and g to functions for which s/he believes the preconditions to be true.

The Law of Averages Method The *law of averages*[3] method uses the fact that, on average, the proportion of time an object has a given property is equal to the proportion of objects in a larger population with that property at a given time. This can be formalised as:

$$S \neq \emptyset \wedge T \neq \emptyset \implies \frac{\|t \in T | \phi(\epsilon S, t)\|}{\|T\|} \rightsquigarrow \frac{\|s \in S | \phi(s, \epsilon T)\|}{\|S\|} \tag{3}$$

For example, the proportion of time an average person spends asleep is equal to proportion of people on Earth asleep at any time, where S is the set of people, T is a finite set of equal time intervals in a day, and $\phi(s,t)$ asserts that person s is asleep during time interval t.

The Distance Method The *distance* method is a domain-specific technique for calculating the distance between two locations on Earth. It applies in the case of a problem such as, *How much time would it take to drive from London to Manchester?*, where two locations are given and a distance is required. *distance* calculates an exact value using the following formula. For two points $\langle \phi_s, \lambda_s \rangle$ and $\langle \phi_f, \lambda_f \rangle$, where the ϕs represent latitudes and λs represent longitudes, the planar angle between the points is calculated[4] by the formula:

$$\Delta\widehat{\sigma} = 2 \arcsin \left(\sqrt{\sin^2 \left(\frac{\Delta\phi}{2} \right) + \cos \phi_s \cos \phi_f \sin^2 \left(\frac{\Delta\lambda}{2} \right)} \right).$$

Then the distance along the surface of the Earth is $r \cdot \Delta\widehat{\sigma}$, where the single significant digit approximation of r, the radius of the Earth, is 6×10^4 km.

[3] This name is adopted from [13]. The normal pejorative use of this phrase is not intended.

[4] `http://en.wikipedia.org/wiki/Great-circle_distance`

2.2 Secondary Methods

The Arbitrary Object Method The *arbitrary object* method uses Hilbert's ϵ operator to convert the value of some function of a set into the value of that function on a typical member of that set. This can be formalised as:

$$f(S) \rightsquigarrow f(\epsilon S) \tag{4}$$

For example, S might be the set of humans and $f(s)$ the height of the human s.

The Average Value Method The *average value* method guesstimates a numeric value for some $f(\epsilon S)$ by computing the arithmetic mean of all $f(s), s \in S$:

$$S \neq \emptyset \implies f(\epsilon S) \rightsquigarrow \frac{\sum_{s \in S} f(s)}{\|S\|}. \tag{5}$$

An application of this method could be to find the average runtime of a typical film, based on knowledge about runtimes of particular films. In this case, S would be the set of all films and $f(s)$ the runtime of the film s.

The Aggregation over Parts Method A guesstimation problem may require a quantity for a large object that is composed of many non-overlapping smaller objects. This is formalised as the *aggregation over parts* method:

$$f(o) \rightsquigarrow \sum_{p \in Parts(o)} f(p), \tag{6}$$

where $Parts(o)$ is a function that returns the set of all non-overlapping parts of o. One such example could be a need for the population of a continent. In that case, the continent could be subdivided into non-overlapping regions, such as countries. o would be the continent and each p a country in that continent. The population of o would be calculated as the sum of the populations of the ps.

The Generalisation Method The *generalisation* method finds more general information when it isn't available for the typical member of a specific set. By looking at the properties of the typical member of a superset, an approximate value can be found.

$$S \subset T \implies f(\epsilon S) \rightsquigarrow f(\epsilon T) \tag{7}$$

For instance, suppose we cannot discover the thickness of a typical lottery ticket. Knowing that lottery tickets are made from cardboard, we can seek instead the thickness of a typical piece of cardboard. S would be the set of lottery tickets and T the set of cardboard objects.

The Geometry Methods Guesstimation problems often need to reason about the physical properties of an object, such as its surface area or volume. Where a precise value for the needed measurement is unavailable, it may be possible to calculate the required measurement from other knowledge, for example a sphere's circumference, $Circ(s)$, given its radius, $Radius(s)$. The *geometry* methods require knowledge of the shape of an object and a measurement. An example of a *geometry* method is one that expresses the circumference of a circular object, $Circ(s)$, in terms of its radius, $Radius(s)$:

$$Circ(s) \rightsquigarrow 2\pi Radius(s)$$

Similarly, methods have been implemented for computing the volume and the surface area of both spherical and rectangular prism objects.

The User Interaction Method The solution to a guesstimation question may rely on information that GORT cannot find on the internet. The *educated guess* method then asks the user for the required value. The user can also elect to use this method as a primary one.

2.3 Towards an Axiomatic Equational SingSigDig Theory

The question naturally arises as to whether we can develop an axiomatic equational SingSigDig theory in which the rewrite rules of this section are theorems. We have begun some experiments towards this end to explore some of the options. The domain-specific rewrite rules generally follow just from the rules of algebra, trigonometry, geometry, etc. We, therefore, restricted our attention to the general-purpose rewrite rules. Firstly, we made unoriented versions of each of them, considered as equations over the $=^{\sim}$ relation. Then we considered which of them could be derived from the others.

If equations based on rewrite rules (4) and (7) are adopted as axioms, then equations based first on (1), and then on (5), (2) and (3), can all be proved as theorems. Equation (6) requires a definition of $Parts(o)$ in terms of o, or could itself be adopted as an axiom.

Note that, as oriented, the rewrite rules for the *arbitrary object* (4), *average value* (5) and *total size* (1) methods have the potential to loop. Such loops are currently prevented by the division of the proof plan into separate primary and secondary phases. The potential loops we have identified all contain both a primary and a secondary method. Since no primary method is allowed to follow a secondary method, such loops do not arise in practice. If, in the future, this restriction is relaxed, or if purely primary or secondary loops are discovered, then a loop checking mechanism will be required.

GORT's proof methods are approximate in two senses. Not only do they return an answer only accurate to within a single significant digit, but they are also fallible. If, for some function $f^{\sim}$ and set S the Similarity Assumption 1 is violated then different ways of evaluating $f^{\sim}(\ldots, S, \ldots)$ can return different values. Currently, it is the responsibility of the user to check that this assumption

is met. We would like to automate this check, but it seems inherently resistant to automation.

An anonymous referee wondered whether it might be possible to order the methods by some fallibility measure, e.g., the probability of their truth, and use this measure during search control. Given their interderivability, it seems unlikely that it will be possible to order the proof methods in this way. Rather, fallibility is not due to the particular method used, but rather to the violation of the Similarity Assumption for some function and set.

3 Worked Examples

We now illustrate how GORT can combine the proof methods from §2 into a proof plan to solve a guesstimation problem. Our worked example is taken from [13].

Problem: *How many golf balls would it take to circle the Earth at the equator?*

Solution: We begin by identifying the type of plan that is appropriate for this question. The result of the question must be a count of a set of golf balls; therefore, the appropriate proof plan is the *count* method described in §2.1. The objects being considered are the set of golf balls required, $Golf\_Balls$, and the object, $earth$[5], and the properties under consideration are the diameter and circumference of these objects, respectively.

We start by choosing the *count* method (2). The user instantiates big, $Small$, f and g to $earth$, $Golf\_Balls$, $Circ$ and Dia, respectively. The user also confirms that the preconditions $Dia(Golf\_Balls) \neq 0$ and $Circ(earth) = \sum_{s \in Golf\_Balls} Dia(s)$ are satisfied. This creates the following rewrite rule:

$$\| Golf\_Balls \| \rightsquigarrow \frac{Circ(earth)}{Dia(Golf\_Balls)} \tag{8}$$

whose RHS must be evaluated to provide the required value for the LHS.

Since $Golf\_Balls$ is a set, the *arbitrary object* method applies and rewrites the RHS of (8), giving:

$$\| Golf\_Balls \| \rightsquigarrow \frac{Circ(earth)}{Dia(\epsilon Golf\_Balls)}. \tag{9}$$

Continuing, we now need values for $Circ(earth)$ and $Dia(\epsilon Golf\_Balls)$. We take an *educated guess* for the diameter of a golf ball:

$$Dia(\epsilon Golf\_Balls) \rightsquigarrow 4.10^0 \text{ cm}$$

Next, we need the circumference of the $earth$. [13] uses background knowledge about flights and time zones to guesstimate the circumference. However, with an information retrieval system at our disposal, we can easily look up the radius

[5] Recall the upper/lower case convention for sets/objects given in §1.2.

of the *earth*, calculate the circumference and rewrite the units to match those of our archetypal golf ball:

$$\begin{aligned} \mathit{Radius}(\mathit{earth}) &\rightsquigarrow 6.10^3 \text{ km} \\ \mathit{Circ}(\mathit{earth}) \rightsquigarrow 2\pi \times \mathit{Radius}(\mathit{earth}) &\rightsquigarrow 4.10^5 \text{ km} \rightsquigarrow 4.10^9 \text{ cm} \end{aligned}$$

Finally, we continue the plan from (9) to obtain the result:

$$\frac{\mathit{Circ}(\mathit{earth})}{\mathit{Dia}(\epsilon \mathit{Golf\_Balls})} \rightsquigarrow \frac{4.10^9 \text{ cm}}{4.10^0 \text{ cm}} \rightsquigarrow 1.10^9 \qquad \square$$

4 Implementation

GORT is implemented as a collection of modules, each of which is described below.

4.1 Basic Ontology

GORT needs background knowledge to determine the appropriate proof plans that apply at a given point in a guesstimation solution. The basic ontology consists of an upper ontology and a small set of ground facts. The knowledge base encompasses the following bodies of knowledge.

- shapes and geometric properties of objects, such as roundness, whether an object is tangible, etc;
- measurement units and dimensions, to support scale unit conversion and common conversion factors;
- a hierarchy of concepts and entailments that allows reasoning about subsumption relationships between sets (e.g., all *Actor*s are also *Person*s);
- a large range of instances of sets, to allow the user to express questions on a broad number of topics.

4.2 Inference System

The proof plans, composed of the methods described in §2, form the basis of GORT's ability to reason over the facts in the knowledge base. The system implements both general and domain-specific methods, and refers to knowledge in the knowledge base to select an appropriate method. Primary methods are selected by the user and secondary method are selected automatically by exhaustive, depth-first rewriting. The proof planner handles failure by backtracking to attempt other plans when possible. If none of the proof plans achieve a result, then the planner will need to trigger a user interaction to get a needed fact.

4.3 The Web Interface

A prototype web interface has been developed for GORT. It uses AJAX to load the results asynchronously. A drag-and-drop interface (using the BBC's GLOW JAVASCRIPT LIBRARY [6]) allows users to select primary methods and provide the parameters required by these methods. When a proof method receives these parameters, it updates itself and any other methods which rely on its output.

4.4 Customised Ontology

The Customised Ontology serves two purposes. First, it provides a mechanism for GORT to record intermediate calculation results and facts that it has retrieved from either the knowledge base or some outside source. Second, because the custom ontology records each intermediate calculation and retrieved fact, it also makes explicit the knowledge that GORT uses to solve a guesstimation question.

4.5 Information Retrieval

GORT needs a way to gather data in response to a user query, and combine it with the proof plans and background knowledge already in the knowledge base to arrive at a solution. This module is responsible for gathering appropriate information in response to a user query. It will eventually be adaptable to a range of information sources, such as Semantic Web sources and natural-language knowledge sources.

GORT, currently supports two methods of information retrieval. Firstly it has various pre-stored ontologies in the form of RDF triples, with explicit links into them using `rdfs:subClassOf`, `rdfs:subPropertyOf`, and `owl:sameAs` relations. Secondly, it uses SPARQL to dynamically link to ontologies that have an appropriate endpoint.

SPARQL(SPARQL Protocol and RDF Query Language) is an RDF query language. It allows the querying of RDF datastores, which means that only relevant data is returned. This avoids having to download lots of unnecessary data, which could be a problem for large datastores. There are a range of datastores with SPARQL endpoints[7], including DbPedia[8], the BBC[9] and Edubase[10]. Queries are sent over HTTP to the SPARQL endpoint, which then returns the relevant results. GORT uses parts of the CLIOPATRIA SEMANTIC SEARCH LIBRARY[11] for SWI-Prolog to assist in querying the endpoints.

Another benefit of using SPARQL is that GORT doesn't need to remember URIs for every object. Queries can be sent using the English labels given by the user and numerical results are returned.

[6] `http://www.bbc.co.uk/glow/`
[7] `http://esw.w3.org/topic/SparqlEndpoints`
[8] `http://dbpedia.org/sparql`
[9] `http://api.talis.com/stores/bbc-backstage/services/sparql`
[10] `http://services.data.gov.uk/education/sparql`
[11] `http://e-culture.multimedian.nl/software/ClioPatria.shtml`

5 Evaluation

The results of a run of GORT on 8 example test problems are summarised in Table 1. The tests were run on a Linux-based, 3Ghz HP dc7900 running SWI-Prolog version 5.8.1. Problem 2 was run twice — once without the data for Loch Ness (2*), and again after having loaded it.

	Problem	Answer	Target	User	Time (s)
1.	*How many cells are there in the human body?*	2.10^{14}	1.10^{14}	$\surd$	10.4
2*.	*How many golf balls would it take to fill Loch Ness?*	*fail*	*fail*	n/a	< 0.1
2.	*How many golf balls would it take to fill Loch Ness?*	2.10^{14}	1.10^{14}	$\surd$	< 0.1
3.	*If all Europeans were placed head-to-toe, how far would they reach?*	1.10^{9} m	1.10^{9} m	$\times$	11.0
4.	*How many people would be needed to form a chain from central London to central Edinburgh?*	3.10^{5}	3.10^{5}	$\times$	10.4
5.	*How many Loch Nesses would fit into the Red Sea?*	3.10^{4}	3.10^{4}	$\times$	0.17
6.	*How many Hangzhou Bay Bridges would it take to cross the Doppler crater on the moon?*	3.10^{0}	3.10^{0}	$\times$	0.17
7.	*If everyone in the crowd at Croke Park drove a Volkswagen New Beetle and parked them in straight line, how long would the line be?*	3.10^{8} mm	3.10^{8} mm	$\times$	0.5
8.	*How many Channel Tunnels would it take to stretch from Edinburgh to New York?*	2.10^{2}	1.10^{2}	$\times$	0.4

The Answer *column gives* GORT*'s answer and* Target *gives the target result. A* $\surd$ *in the* User *column shows that user input was required via the* educated guess *method; a* $\times$ *shows it wasn't. The* Time *column shows the average CPU time in seconds, averaged over 10 runs.*

Table 1. Results for all Test Problems.

The 'Target' results are taken from third party sources, where available, such as [13]. Note that GORT produced results accurate to within a single significant digit for 5 of the 8 problems and within an order-of-magnitude for the other three. The three discrepancies arise from differences in the proof plans used by GORT and human guesstimators, and from the inherent fallibility of GORT's guesstimation methods (see discussion in §2.3).

To estimate the success rate of GORT it was run on all 11 of the 'general questions' from [13][Chap. 3]. These 11 problems were chosen because they were a wide and representative sample from an independent source. GORT was able to solve 6 of these 11 problems. For all 6 successful problems it returned a result identical to that given by Weinstein and Adams. The remaining 5 failed for various reasons. 2 failed because GORT did not have a guesstimation method

able to solve rate of change problems (see §7.2 for further discussion). The other 3 failed due to the lack of the required geometric or chemical knowledge.

The adaptability of the system was assessed by its ability to modify its execution plan when new RDF triples appeared in the knowledge base, simulating the appearance of new knowledge in the web of linked data. In problem 2, when the volume data for Loch Ness was missing, GORT queried the user to provide it, but the user declined, so the attempt failed. The missing information was then provided and the problem successfully rerun.

6 Related Work

There are no systems that are directly comparable to GORT. However, there are several systems that share common characteristics. In this section, we compare GORT with five other Semantic Web and knowledge-based systems: PowerAqua, a Semantic Web-based question answering system [7], QUARK, a domain-independent, logic-based, natural-language, question-answering system [12], CS Aktive Space, a system for tracking UK computer science research [11], Cyc, a general-purpose 'common-sense' reasoning system [6], and Wolfram|Alpha, a system that calculates answers to numerical questions on a wide range of topics[12]. The comparisons will be conducted along the four dimensions below, which were proposed in [9] for evaluating next-generation Semantic Web applications. They have been used to compare other Semantic Web systems, so act as an objective set of evaluation criteria.

1. the system's ability to re-use Semantic Web data;
2. whether the system is *single-ontology* or *multi-ontology*;
3. the system's ability to adapt to new Semantic Web resources at the request of the user; and
4. the system's ability to scale.

Data Reuse To begin, we consider the systems' abilities to reuse data from other Semantic Web systems. The earliest system under consideration is Cyc, which is a large-scale, curated, knowledge base that is not able to directly incorporate knowledge from Semantic Web sources, although there are techniques for mapping Cyc concepts to external concepts. CS Aktive Space (CSA) was designed in the early days of the Semantic Web, and thus little data was available. It is not designed to dynamically adapt to other Semantic Web resources until their contents have been translated into its AKT reference ontology. Wolfram|Alpha is the newest of the systems under consideration. Like Cyc, it uses a closed, hand-curated data set to perform its inferences and does not incorporate a facility for accessing Semantic Web resources. PowerAqua, QUARK and GORT, on the other hand, are each designed to operate with data from other Semantic Web resources. PowerAqua and QUARK can answer questions based on data gathered from a large number of ontologies, and do so dynamically at runtime. GORT is also capable of incorporating data from several Semantic Web data sources.

[12] `http://www.wolframalpha.com/`

Single- or Multi-Ontology Secondly, we consider whether each system is *single-ontology* or *multi-ontology*. This distinction is important: only a *multi-ontology* system assumes that it operates in a larger data ecosystem such as the Semantic Web. As systems with hand-curated, proprietary knowledge bases, both Cyc and Wolfram|Alpha are clearly single-ontology systems. These systems do not appear to be capable of working with more than their own ontology. Although CS Aktive Space incorporates data from multiple sources, all of its data is re-mapped into the AKT reference ontology. As previously mentioned, PowerAqua and QUARK can work with multiple ontologies at the same time. PowerAqua discovers and integrates these at runtime. QUARK is linked to various information interchange sources. This flexibility makes it particularly easy to incorporate new Semantic Web data sources into PowerAqua and QUARK — they have been designed with multiple ontologies in mind. GORT is also capable of working with multiple ontologies. The use of SPARQL makes it fairly easy to add new ontologies by adding their endpoints.

Openness Thirdly, we consider each system's openness to new semantic resources. Three of the systems are not especially amenable to the incorporation of new semantic resources: none of Cyc, Wolfram|Alpha, or CS Aktive Space can incorporate new RDF content in response to a user query. This is a consequence of each of those systems' single-ontology approach. There are techniques for mapping new ontologies into Cyc, but these ontologies cannot automatically be retrieved and mapped at the user's request. PowerAqua is capable of incorporating new ontologies into its query answering at its user's request, without additional configuration. QUARK can be readily reprogrammed to link it to additional ontologies. GORT is also open to new ontologies via its SPARQL interface. By adding an ontology's SPARQL endpoint, its data becomes available.

Scalability Finally, we consider each system's ability to scale to large data sets. Although Cyc's knowledge base is large and supports complex inferences, it is small in comparison to the projected size of the Semantic Web. Although Cyc is adaptable to Semantic Web sources, no testing has been done to evaluate its performance on large, non-curated data sets. Similarly, QUARK is linked to some very large ontologies, but we could find no experimental data on its scalability. There is no public estimate of the amount of data stored in Wolfram|Alpha, but the range of problems for which it provides an answer suggests a very large knowledge base, but this is unlikely to approach the magnitude of the Semantic Web. PowerAqua is designed with the large-scale Semantic Web in mind, and its query performance has been tested against large data sets. The PowerAqua system has access to a larger data set, more sophisticated inference algorithms, and is capable of answering a larger variety of question types than GORT, although it does not solve guesstimation problems. The 0–11 second response time of GORT with a database of 3 million triples, suggests that it is scalable up to several million more, still staying within tolerable response times.

7 Conclusion

In this paper we have argued that proof planning in an SingSigDig calculus can be successfully used to solve guesstimation problems. We have implemented this technology in the GORT system and applied it to a representative sample of guesstimation problems §5. GORT has been compared favourably with related systems in §6.

7.1 Discussion

We now consider the success of GORT by the criteria defined in §1.3, namely: the proportion of accurate results returned from its test set; its adaptability; and its efficiency.

In §5, GORT was able to produce an answer to all 8 of the test set of guesstimation problems. On 5 problems, GORT guesstimated the target single significant digit answer. On 3 other problems it came within an order-of-magnitude of the target answer. These discrepancies arose from different choices in the proof plans used to guesstimate the answers, and seem inevitable given the inherent approximate nature of guesstimation. On another run, designed to evaluate its success rate on an independently sourced test set, GORT successfully guesstimated 6/11 problems to within a single significant digit.

GORT was shown to be adaptable via the experiments on problem 2, as discussed in §5. Further such experimentation is desirable.

The timing data shown in Table 1 indicate that each query completed in less than 11 seconds, over a knowledge base of approximately 3 million RDF triples. Profiling shows that the system spends most of its time in tactics that perform list aggregation, such as the *average value* and *aggregation over parts* plans. We have also investigated the computational complexity of GORT's various proof methods. *average value* and *aggregation over parts* both use breadth-first search, and so have complexities $O(b^d)$, where b is the branching rate and d is the depth of the search space. The other secondary methods all have complexity $O(1)$; the complexity of the primary methods depends on the secondary methods they call.

7.2 Further Work

To extend GORT to handle the full range of guesstimation problems found in sources such as [13, 10] requires several additional methods. For instance, many problems require examining rates, such as "How long would it take to fill the dome of St. Paul's Cathedral with water from a typical garden hose?" To solve this problem would require a proof method for reasoning about rates of flow.

To make GORT easier to use, the web-service interface needs considerable improvement. In the long term, we plan to build a natural language interface, so that guesstimation problems can be posed as English questions. We also plan to automate the choice of top-level proof method, by analysing the form of the question. This would include, for instance, automating the instantiation and

checking of the preconditions of methods such as *count*[13]. We will also continue to explore the development of an axiomatic equational theory for the SingSigDig calculus, as discussed in §2.3.

References

1. Jonathan A. Abourbih. Method and system for semi-automatic guesstimation. Master's thesis, University of Edinburgh, Edinburgh, Scotland, August 2009.
2. L. Blaney. Semi-automatic guesstimation. University of Edinburgh, Undergraduate Project Dissertation, 2010.
3. A. Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.
4. A. Bundy and M. Chan. Towards ontology evolution in physics. In W. Hodges, editor, *Procs. Wollic 2008*. LNCS, Springer-Verlag, July 2008.
5. David Hilbert and Paul Bernays. *Die Grundlagen der Mathematik — Zweiter Band*. Number L in Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen. Springer, 1939.
6. D.B. Lenat. CYC: a large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, November 1995.
7. Vanessa Lopez, Davide Guidi, Enrico Motta, Silvio Peroni, Mathieu d'Aquin, and Laurian Gridinoc. Evaluation of semantic web applications. OpenKnowledge Deliverable D8.5, Knowledge Media Institute, The Open University, Milton Keynes, England, October 2008. Accessed 10 August 2009.
8. F. McNeill and A. Bundy. Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *International Journal On Semantic Web and Information Systems*, 3(3):1–35, 2007. Special issue on ontology matching.
9. Enrico Motta and Marta Sabou. Next generation semantic web applications. In *The Semantic Web – ASWC 2006*, pages 24–29. Springer, Berlin, 2006.
10. A. Santos. *How Many Licks: Or, How to Estimate Damn Near Anything*. Perseus Books, 2009.
11. N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and M.C. Schraefel. CS AKTive space, or how we learned to stop worrying and love the semantic web. *IEEE Intelligent Systems*, 19(3):41–47, 2004.
12. R. Waldinger, J. Hobbs, D.E. Appelt, J. Fry, D.J. Israel, P. Jarvis, D. Martin, S. Riehemann, M.E. Stickel, M. Tyson, and J.L. Dungan. *New Directions in Question Answering*, chapter Deductive question answering from multiple resources, pages 253–262. AAAI Press., 2003.
13. L. Weinstein and J.A. Adam. *Guesstimation: solving the world's problems on the back of a cocktail napkin*. Priceton University Press, 2008.

[13] This will be the topic of a new MSc project by Aparna Ghagre.