Scalable Knowledge Harvesting with High Precision and High Recall

Ndapandula Nakashole, Martin Theobald, Gerhard Weikum
Max Planck Institute for Informatics
Saarbrücken, Germany
{nnakasho,mtb,weikum}@mpi-inf.mpg.de

ABSTRACT

Harvesting relational facts from Web sources has received great attention for automatically constructing large knowledge bases. State-of-the-art approaches combine pattern-based gathering of fact candidates with constraint-based reasoning. However, they still face major challenges regarding the trade-offs between precision, recall, and scalability. Techniques that scale well are susceptible to noisy patterns that degrade precision, while techniques that employ deep reasoning for high precision cannot cope with Web-scale data.

This paper presents a scalable system, called PROSPERA, for high-quality knowledge harvesting. We propose a new notion of n-gram-itemsets for richer patterns, and use MaxSat-based constraint reasoning on both the quality of patterns and the validity of fact candidates. We compute pattern-occurrence statistics for two benefits: they serve to prune the hypotheses space and to derive informative weights of clauses for the reasoner. The paper shows how to incorporate these building blocks into a scalable architecture that can parallelize all phases on a Hadoop-based distributed platform. Our experiments with the ClueWeb09 corpus include comparisons to the recent ReadTheWeb experiment. We substantially outperform these prior results in terms of recall, with the same precision, while having low run-times.

Categories and Subject Descriptors

H.1.0 [**Information Systems**]: Models and Principles—*General*

Keywords

Knowledge Harvesting, Information Extraction, Scalability

1. INTRODUCTION

1.1 Motivation

Major advances in information extraction [25, 14] and the success and high quality of knowledge-sharing communities like Wikipedia have enabled the automated construction of large knowledge bases [1, 32]. Notable efforts along these lines include ground-breaking academic projects such as *opencyc.org*, *dbpedia.org* [4], *knowitall* [16, 5], *stat-snowball* [37], *readtheweb* [9], and *yago-naga* [29], as well as commercial endeavors such as *wolframalpha.com*, *freebase.com*, and *trueknowledge.com*. Such knowledge bases contain millions of entities and hundreds of millions of facts about them. DBpedia and YAGO harvest Wikipedia categories, infoboxes, and lists, achieving very high precision (fraction of correct facts). However, there are inherent limitations regarding recall (extent of facts captured). Most interesting facts are expressed only in the natural-language text of Wikipedia articles, which is much harder to comprehend by a machine; and even more is available in other sources such as scholarly publications, news sites, and personal or business Web pages.

Therefore, it is crucial to tap on *natural-language* sources for further knowledge harvesting, and to continuously repeat this effort. Unfortunately, the methods for doing this are much less robust and incur much higher computational costs. Shallow parsing with pattern-based extraction has high recall and is fast, but yields noisy results with mediocre or worse precision. Deep parsing (e.g., dependency parsing) and constraint-based reasoning (e.g., Markov logic networks), on the other hand, are expensive and do not easily scale to Web dimensions. Moreover, they tend to be overly conservative and miss out on recall. We are thus facing a threefold trade-off between precision, recall, and scalability. This paper aims to reconcile all three of these goals.

1.2 State of the Art and Open Problems

Prevalent approaches to automatic knowledge harvesting roughly fall into two categories: *pattern-based extractors* and methods that employ *consistency constraint reasoning* (or some form of constraint-aware statistical learning).

In a pattern-based system (e.g., [6, 2, 5, 8]), *seed facts* like *(Germany, FIFA_World_Cup)*, as an instance of the *teamWonTrophy* relation between soccer teams and trophies (which was true in 1974 and 1990), can automatically detect textual *patterns* like "X won the final and became the Y champion" which in turn can discover new facts such as *(Spain, FIFA_World_Cup)* (which is true for 2010). While this is good for high recall, it may lead to noisy patterns and degrading precision. For example, the same seed may lead, after a few iterations, to frequent patterns such as *X lost the final of Y* and erroneously pick up the false positive *(Netherlands, FIFA_World_Cup)*.

Reasoning-enhanced systems (e.g., [30, 37, 9]) check the plausibility of the extracted *fact candidates* by their mutual consistency based on specified logical constraints. For example, when a system generates a candidate *(Tim_Berners-Lee, Max_Planck)* for the *hasAcademicAdvisor* because of a spurious pattern *X benefitted from the deep insight of Y*, a constraint-aware system can invalidate this hypothesis by using prior knowledge about the birth year of Berners-Lee (1955) and the death year of Planck (1947) and a rule that a

WSDM'11, February 9–12, 2011, Hong Kong, China.

student must be born (or at least, say, 15 years old) before graduating. Similarly, type constraints can be used to eliminate false positives such as *(Tim_Berners-Lee, Chelsea_Football_Club)* for the *graduatedFrom* relation.

Consistency reasoning is expensive and faces *scalability problems*. The *ReadTheWeb* project [9, 10] recently showed how to parallelize a constraint-aware approach, using an ensemble-learning method over coupled pattern learners. However, this method yields relational pairs between *names* of entities rather than referring to *entities* themselves. For example, the result may contain instances such as *(Real_Madrid_CF, Champions_League), (Real_Madrid, UEFA), (Real, UEFA_Champions_League)* – three instances that refer to the very same fact (the experiment in [10] reports on American sports, e.g., returning *Yankees* and *New York Yankees* as if they were different entities). This is much lower-quality knowledge than a canonical representation that would state the fact in terms of uniquely named entities – *(Real_Madrid_CF, UEFA_Champions_League)* – with a separate relation for capturing the synonyms of entities (e.g., *means(Real, Real_Madrid_CF)*. Note that many names are ambiguous, so they appear in *means* facts for different entities. For example, Wikipedia knows more than 20 soccer clubs called Real, there are several currencies Real, and so on.

The SOFIE system [30], on which the current paper builds, combines a *pattern-gathering phase* with a *consistency-reasoning phase*. Experiments in [30] dealt with 1,000's of input documents and 10,000's of hypotheses in about 10 hours on a single computer. Clearly, this is not good enough for Web scale. Moreover, while precision was very high, the constraint-based reasoning severely limited recall. To increase recall, one could relax the pattern gathering and allow more patterns and co-occurring fact candidates to enter the reasoning phase. However, this entails two problems: 1) *efficiency:* it presents the reasoner with a much larger hypotheses space (more clauses for a Weighted-MaxSat solver [30]); 2) *quality:* it introduces noisy patterns such as "X played in the qualification round of Y" for *teamWonTrophy*, and sparsely occurring patterns such as "P scored her third goal in the triumph of T" for the *athletePlaysForTeam* relation, which is hardly generalizable for learning given that female players have less coverage in Web sources and *"three"* (goals in the same match) is a rare feature.

In summary, Web-scale knowledge harvesting is not yet as robust and scalable as we wish it to be. Key issues are:

- *non-canonical output:* shallow and scalable methods yield pairs of (ambiguous) names rather than truly entity-based facts;
- *scalability:* deep methods with constraint-based reasoning do not scale and face efficiency challenges;
- *limited recall:* reconciling high precision with high recall is challenging because of the need for and difficulty in dealing with noisy and sparse patterns.

1.3 Contribution

This paper overcomes, to a large extent, the above problems and shows how to reconcile high precision, high recall, and scalability. Our solution, coined PROSPERA (PRospering knOwledge with Scalability, PrEcision, and RecAll), is based on the SOFIE framework [30], but includes major extensions along the following lines.

For pattern-based gathering of fact candidates, we introduce a new notion of *n-gram-itemset patterns*. In prior work, patterns are either consecutive substrings from the surface text that connects two named entities in a document or words on the path of the linkage graph that results from dependency-parsing the text. The former is noisy, the latter entails an expensive deep-parsing step. To gather more interesting patterns in an inexpensive way but avoid becoming even noisier, we define a pattern to be a set of variable-length n-grams in the context of two entities, and we employ algorithms for frequent-itemset mining to efficiently compute pattern statistics. Moreover, we allow patterns to lift individual parts into their corresponding part-of-speech tags (word categories); the resulting generalized patterns have higher support. For example, the lifted n-gram-itemset *P* { *scored PRP ADJ goal; match for* } *T* covers all sentences with the two phrases regardless of which pronouns (PRP) or adjectives (ADJ) are used and regardless of any preceding, following, or intermediate text such as relative clauses or appositions.

For constraint-based reasoning, we leverage SOFIE's Weighted-MaxSat solver, but use confidence measures from pattern gathering for setting clauses weights. We also consider negative patterns that often co-occur with entity pairs that are not in the relation of interest. For example, if we specify upfront that *(Netherlands, FIFA_World_Cup)* is not an element of the *teamWonTrophy* relation, then we can automatically detect that *"lost the final"* is a negative pattern; this will in turn downgrade the weights of clauses that relate this pattern with fact candidates. As the reasoning still prioritizes precision over recall, the entire two-phase procedure is iterated, with re-adjusted weights, yielding much higher recall than the original SOFIE.

Finally, to gear up for Web-scale harvesting, we have developed a new architecture where tasks of all phases can be distributed across many computers and run in parallel. For the pattern gathering and analysis, we partition by input sources, but ensure that the n-gram-itemset mining benefits from global statistics. For the reasoning, we employ graph-partitioning algorithms on the clauses graph and construct parallel reasoning tasks. Tasks are mapped to distributed machines using the Hadoop software.

In summary, the paper's novel contributions are:

- *rich and accurate patterns:* a new notion of n-gram-itemsets to generalize narrow patterns and more precisely capture noisy patterns;
- *clauses weighting:* using confidence and support statistics from the pattern-based phase for carefully weighting the reasoner's input clauses and re-weighting them in iterations of both phases;
- *distributed architecture:* organizing both phases in a way that data and load can be partitioned across parallel machines for better scalability;
- *Web-scale experiments:* demonstrating the run-time efficiency, high precision, and high recall of our methods by performing knowledge harvesting on the ClueWeb09 corpus (with 500 million Web pages, *boston.lti.cs.cmu.edu/Data/clueweb09/*), comparing ourselves to the ReadTheWeb experiments of [10], and outperforming their results by a large margin.

The n-gram-itemsets patterns have been introduced in preliminary form in our short workshop paper [21]. Here we show how to scale this approach up to Web proportions. The other contributions are completely new. All data on the experiments reported here are made available as supplementary material on the Web site *www.mpi-inf.mpg.de/yago-naga/prospera/*.

2. RELATED WORK

Declarative approaches to information extraction, such as System T [22], Cimple [26], or SQOUT [17] orchestrate pattern-matching steps into execution plans which allow database-style optimizations. However, they are limited to deterministic patterns like regular

expressions and are not geared for dealing with the inherent uncertainty of full-fledged natural language.

Pattern-based fact gathering (e.g. [6, 2, 12, 16, 5, 7, 36, 8]) is bootstrapped with seed facts for given relations and automatically iterates, in an almost unsupervised manner, between collecting text patterns that contain facts and finding new fact candidates that co-occur with patterns. Statistical measures such as PMI are used to assess the goodness of newly found facts and of characteristic patterns. This is a powerful machinery for high recall, but it often leads to noisy patterns and may easily drift away from the target relations.

Statistical-learning methods for relational graphs (e.g. [23, 15, 11, 37, 9, 24]) aim to overcome these problems by incorporating probabilistically weighted rules for coupling the random variables that denote whether fact candidates are true or false, and then using joint inference over all fact candidates together. This is a powerful for higher precision, but it tends to reduce recall and has high computational cost as inference is typically based on Markov-chain Monte-Carlo methods such as Gibbs sampling. Note that large-scale knowledge harvesting can easily lead to situations with hundred thousands of highly intertwined random variables.

The *ReadTheWeb* project [9] has recently shown how to parallelize such a method for large-scale knowledge gathering, using an ensemble-learning approach with coupled pattern learners. In the NELL experiment [10], it produced about 230,000 facts for 123 different categories (unary relations) and 12,000 facts for 55 different types of binary relations from a large Web corpus with 500 million pages (a variant of the ClueWeb09 corpus). Although this is the largest-scale experiment of this kind in the literature, it does have notable limitations. First, its recall on binary relations - our primary focus - was rather low: only 12,000 facts after 66 days, and quite a few of the 55 relations acquired less than 100 facts. Note that one could retrieve the instances of unary relations (categories) much more easily from existing knowledge bases such as DBpedia or YAGO; the real difficulty lies in binary relations. Second, the experiment involved some human interaction on a daily basis, after each iteration of rule learning, to manually remove bad extraction rules. Third, the output refers to non-canonical names rather than uniquely identified entities. The entire experiment ran more than 60 days on a large cluster. In the current paper, we use the ReadTheWeb results (which – thanks to the authors – are available on the Web) as a yardstick against which we measure our approach.

Reasoning-based methods with prior knowledge such as SOFIE [30] leverage knowledge about entities and their types. SOFIE decomposes the entire fact harvesting into two phases: 1) pattern-based gathering of candidates for high recall, 2) reasoning about candidates and constraints for high precision. In contrast to the above mentioned learning methods, there is no probabilistic interpretation over "possible worlds"; instead all hypotheses are organized into a set of propositional-logic clauses with informative weights derived from pattern statistics. Then a customized approximation algorithm for Weighted-Maximum-Satisfiability (MaxSat) is employed to identify a subset of fact candidates and characteristic patterns that together constitute "the truth". Constraints include functional dependencies, inclusion dependencies, type constraints, and more. Type information is extremely beneficial in early pruning of the search space of the MaxSat solver; for example, the first argument of the *teamWonTrophy* relation must be of type *sportsTeam*, as opposed to say *businessEnterprise* (which immediately rules out the hypotheses that Google has won the FIFA World Cup). The mapping of names onto entities can either be encoded into the clauses set, or carried out before the reasoning by using pre-existing knowledge like the YAGO ontology [29] which knows more than 2 million entities organized in ca. 200,000 semantic classes and has a rich repository of synonyms gathered from redirection pages. and hyperlink anchor texts in Wikipedia.

3. PROSPERA ARCHITECTURE

We are given a set of binary relations $R_1, \ldots, R_m$ of interest, each with a type signature and a small set of *seed facts* and, optionally, a set of *counter-seeds*. The latter are entity pairs that are asserted to be definitely not among the instances of a given relation (e.g., *(USA, FIFA_World_Cup)* for the *teamWonTrophy* relation. We assume that we have an existing knowledge base with typed entities, which more or less includes all individual entities and their types (e.g., *footballPlayer*). In our experiments, we use YAGO [29] for this purpose, which provides us with more than 2 million richly typed entities and a fairly complete dictionary of the possible meanings of a surface string. The YAGO *means* relation explicitly maps surface strings onto individual entities. This does not yet resolve any name-entity ambiguity, though (see below).

We now consider a textual corpus, e.g., a set of Wikipedia articles, Web pages, or news articles, as input for harvesting facts about the relations of interest. PROSPERA processes this data in three phases:

1. *Pattern gathering:* This phase identifies triples of the form (e_1, p, e_2) where e_1 and e_2 are any two entities, each occurring in a different noun phrase, and p is a surface string that appears between e_1 and e_2. As sentences contain merely names rather than entities, we determine all possible mappings using the *means* relation of YAGO and then use a disambiguation heuristics based on context similarities. We form a text-window-based bag of words around the name, and a bag-of-words context around each entity that the name could possibly be mapped to. For the latter, we use the type names of the entities in YAGO (which include names of Wikipedia categories to which the entity belongs). The entity whose context has the highest word-level overlap with the name's bag of words is chosen for the name-to-entity mapping.

2. *Pattern analysis:* This phase generalizes the basic patterns from the previous step by transforming them from long, overly specific phrases into sets of n-grams succinctly reflecting the important sub-patterns. We also compute statistics and similarity measures about patterns and use them to generate fact candidates. The pattern analysis is further detailed in Section 4.

3. *Reasoning:* This step complements the statistical evidence of the previous phase with logical plausibility for high precision. The fact candidates are passed to a MaxSat-based reasoner which considers prespecified contraints to ensure mutual consistency of accepted facts and their compatibility with the consistency constraints. The reasoning phase is further detailed in Section 5.

Our experiments demonstrate that the disambiguation heuristic is very powerful and fairly accurate. Employing this name-entity mapping upfront is a departure from the SOFIE architecture, where entity disambiguation was part of the constraint-based reasoning [30] (and similar techniques were also used in other approaches, e.g., based on Markov logic or factor graphs [15, 33]). However, integrating name-entity mapping into consistency reasoning has high computational cost as it leads to a much larger space of clauses and Weighted MaxSat is an NP-hard problem with inherent combinatorial complexity. Our streamlined approach in this paper is efficient and scales very well to Web proportions.

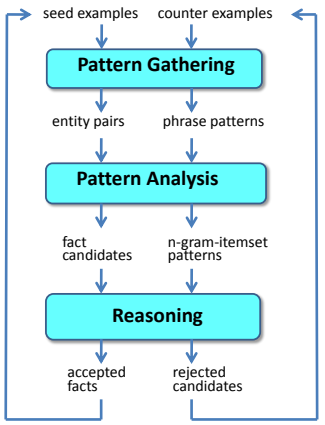

Figure 1: Architecture of the PROSPERA System.

As we will explain later, each of the three phases can be parallelized on a distributed platform. Moreoever, we can iterate the three phases by feeding the output of the reasoner back into the pattern gathering, treating newly found facts as additional seeds for the next iteration. While such feedback loops are well studied for knowledge-harvesting methods that are exclusively pattern-based, such as [6, 2, 5], our approach distinguishes itself from that previous work by including the reasoning phase in each iteration. This strengthens the choice of next-iteration seeds and ensures that precision is kept high. The overall architecture of PROSPERA is illustrated in Figure 1.

4. PATTERN ANALYSIS

4.1 Seed Patterns

The basic pattern phrases from the pattern gathering phase are fed into a frequent *n-gram-itemset* mining algorithm for identifying strong patterns. For example, for the *hasAcademicAdvisor* relation, the pattern gathering may yield a sentence like "*Barbara Liskov* was the first woman in the US who was *honored with* the title of a *doctor of philosophy* (*Ph.D.*) from a technical department at *Stanford University*". Such a long and extremely specific phrase does not generalize, as there are hardly any other entity pairs with exactly the same pattern. A standard technique would be to consider a subsequence as a better pattern, for example, the substring "was honored with the title of a doctor of philosophy". But even this is overly specific and would occur very sparsely in the corpus.

To overcome this problem, we break down the phrases into variable-length n-grams of successive words, i.e., multiple n-grams per phrase. Then a much better pattern would be the *n-gram-itemset* consisting of three n-grams: { honored with; doctor of philosophy; Ph.D. }. A new sentence is a candidate for containing a fact if it contains all three n-grams, not necessarily consecutively and possibly in a different order, or at least a large subset of this n-gram-itemset. For efficiently generating the n-gram-itemset patterns, we apply the technique of frequent itemset mining [3, 28] which has been widely used to discover interesting relations among items in databases.

Often, there is a variety of different wordings regarding pronouns or injected adjectives that render n-grams sparse. For example, because of the variations *"received his"* and *"received her"*, we may dismiss good n-grams as too infrequent. To overcome this issue, we run part-of-speech (POS) tagging on each of the originally gathered sentences, assigning word categories like nouns, verbs, pronouns, etc. to each of the words. We tenatively replace words with their POS tags, to obtain a more general form of n-grams that we refer to as *lifted patterns*. We focus on POS tags for pronouns, prepositions, articles, and adjectives. This way we can generalize the above pattern into the lifted form *"received PRP"* where PRP denotes an arbitrary pronoun.

Definition 1: Given a set $SX(R_i)$ of seed examples for a relation R_i and an input set S of sentences (or, more generally, token sequences), a *basic pattern* p is a sequence such that $e1\ p\ e2$ occurs in S for at least one pair $(e1, e2) \in SX(R_i)$. A *lifted pattern* is a pattern p where all words with certain POS tags are replaced by their tags. An n-gram itemset pattern is a set Q for which there is at least one sequence $s \in S$ that can be written as $s = h\ e1\ p\ e2\ t$ with a seed example $(e1, e2) \in SX(R_i)$ such that for all $q \in Q$ the length of q is at most n words (tokens) and q is a subsequence of p.

To assess the goodness of an n-gram-itemset pattern, we compute the following statistics about co-occurrence of patterns with seed examples and counterexamples. *Support* captures the frequency of a pattern in conjunction with a seed fact, whereas *confidence* reflects the ratio of the pattern co-occurring with seed facts versus counter-seeds.

Definition 2: For sets $SX(R_i)$ and $CX(R_i)$ of seed examples and counterexamples and an input set S of sentences, a basic (or lifted) pattern q has $support(q) =$

$$\frac{|\{s \in S | \exists (e1, e2) \in SX(R_i) : q, e1, e2\ occur\ in\ s\}|}{|S|}$$

and $confidence(q) =$

$$\frac{|\{s \in S | \exists (e1, e2) \in SX(R_i) : q, e1, e2\ occur\ in\ s\}|}{|\{s \in S | \exists (e1, e2) \in SX(R_i) \cup CX(R_i) : q, e1, e2\ occur\ in\ s\}|}$$

Definition 3: An n-gram-itemset pattern q, for given $SX(R_i), CX(R_i)$, and input set S of sentences, is called a *seed pattern* if both $support(q)$ and $confidence(q)$ are above specified thresholds. Pattern q is associated with a seed-pattern weight, set to

$$weight(q) = \alpha \times support(q) + (1 - \alpha) \times confidence(q)$$

In our experiments, we used only confidence values and disregarded support for the weighting ($\alpha = 0$).

4.2 Fact Candidates

The seed patterns are used to discover new fact candidates. We consider all sentences $s \in S$ that contain two entities (x, y) of appropriate types for R_i (e.g., a person and a university for the *hasAcademicAdvisor* relation) and whose subsequence p in $s = h\ x\ p\ y\ t$ in between the two entities x, y partially matches one of the seed patterns. The goodness of the match is quantified by the *Jaccard similarity*

$$sim(p, q) = \frac{|\{\text{n-grams} \in p\} \cap \{\text{n-grams} \in q\}|}{|\text{n-grams} \in p\} \cup \text{n-grams} \in q\}|}$$

This approximate matching of p against all seed patterns q is efficiently implemented by lookups in an n-gram index constructed from the seed patterns.

We process all input sentences $s = h\ x\ p\ y\ t$ this way, and again perform frequent-itemset mining to concentrate on the set of patterns to those with support above a specified threshold. The output of this step is a multi-set of weighted triples $(x, y, p)[w]$ where (x, y) is a fact candidate, p is an n-gram-itemset pattern, and w is the highest pattern-matching similarity of p with any seed pattern q. Note that it is a multi-set rather than a set because the same candidate can be encountered several times.

Definition 4: For given input set S and seed-pattern set Q, the fact-pattern candidate multi-set $C(S,P)$ is:

$$C(S,P) = \{(x,y,p)[w] \mid \exists s \in S : s\ contains\ x,y,p \wedge w = max\{sim(p,q) \times weight(q) | q \in Q\}\}$$

Finally, we can aggregate the fact-pattern candidates in C, grouping them either by fact candidates, to compute a strength measure of the potentially new fact, or by patterns, to quantify the goodness of a pattern. For fact candidates (x,y), the aggregated weight is: $weight(x,y) = \sum\{w|(x,y,p)[w] \in C\}$. For an n-gram-itemset patterns p, the aggregated weight is: $weight(p) = \sum\{w|(x,y,p)[w] \in C\}$.

We can interpret these weights as the statistical evidence that (x,y) is a valid fact and p is a good pattern for further extraction steps for R_i. Note that the two weights are quite different, as the aggregations are computed over different sets.

5. REASONING

Following the work on SOFIE [30], the new PROSPERA system also uses constraints on hypotheses to prune false positives and improve precision. In contrast to [9, 10], we include constraints on the duality of patterns and facts, and we harness the rich knowledge about entity types provided by YAGO. In addition, we specify functional dependencies, inclusion dependencies, relation properties such as symmetry, antisymmetry, or inverse relations, as well as domain-specific constraints whenever possible. The constraints are manually specified upfront, but in all our experiments this was merely a matter of a few minutes. Typical constraints look as follows, with variables $p, e1, e2, e3$ for patterns and entities and given relations R, S, T:

$occurs(p,e1,e2) \wedge type(e1,dom(R)) \wedge type(e2,range(R))$
$\wedge\ expresses(p,R) \Rightarrow R(e1,e2)$ //pattern-fact duality
$occurs(p,e1,e2) \wedge type(e1,dom(R)) \wedge type(e2,range(R))$
$\wedge\ R(e1,e2) \Rightarrow expresses(p,R)$ //pattern-fact duality
$R(e1,e2) \wedge type(R,function) \wedge different(e2,e3)$
$\Rightarrow \neg R(e1,e3)$ //functional dependency
$R(e1,e2) \wedge sub(R,S) \Rightarrow S(e1,e2)$ //inclusion dependency
$R(e1,e2) \wedge inv(R,T) \Rightarrow T(e2,e1)$ //inverse relations
$T(e1,e2) \wedge inv(R,T) \Rightarrow R(e2,e1)$ //inverse relations

For the actual reasoning procedure, the constraints are *grounded* by substituting all meaningful constants – concrete patterns and entities – into the constraint formulas, thus providing us with a set of propositional-logic *clauses*. We can handle clauses with several negative literals, whereas rule-induction methods (e.g., the one used in [9, 10]), are typically restricted to Horn clauses. For example, for the *graduatedFrom* relation (assuming that it were a function: it refers only to Ph.D. degrees and one can obtain a Ph.D. only from one university), the grounding procedure generates clauses such as:

$occurs$(and PRP alma mater, Barbara_Liskov, Stanford_University)
$\wedge\ expresses$(and PRP alma mater, graduatedFrom)
$\Rightarrow graduatedFrom$(Barbara_Liskov, Stanford_University)
$graduatedFrom$(Barbara_Liskov, Stanford_University)
$\Rightarrow \neg graduatedFrom$(Barbara_Liskov, UC_Berkeley)
$graduatedFrom$(Barbara_Liskov, UC_Berkeley)
$\Rightarrow \neg graduatedFrom$(Barbara_Liskov, Stanford_University)

Note that the grounding already evaluates predicates that have only constants as arguments. For example, the *different* predicate between two entities is directly set to true or false, thus simplifying the resulting clauses. Most importantly, the type predicates are evaluated at this stage, too. For entities that do not obey the type signature of the relation at hand, the antecedent of the clause evaluates to false so that the entire clause can be eliminated. This massive pruning of clauses from the hypotheses space greatly reduces the reasoner's load. The efficiency gain is possible because of the rich type information about entities that YAGO provides. Such optimizations were not possible in earlier work on reasoning-based information extraction such as [15]. The recent work of [9, 10] considered type information, too, but only in the form of coupling the pattern-based learners for binary relations with those of unary ones. Note, however, that this was at the level of non-canonical (often ambiguous) names rather than uniquely identified entities; so it is not as clean and powerful as our rigorous typing at the entity level.

The grounded clauses are weighted, and then we finally run the Weighted-MaxSat reasoner of [30]. This computes truth values for all hypotheses on the $expresses$ and $R(.,.)$ predicates for all relations R and all instantiated constants (patterns and entities), such that the total weight of the clauses that are satisfied by this truth-value-assignment becomes as large as possible. The algorithm can only approximate the maximum of this objective function, given that MaxSat is NP-hard and our algorithm runs on hundred thousands of clauses with ten thousands of variables (= fact/pattern hypotheses).

The weights of clauses are derived from the pattern-confidence measures computed in the pattern analysis phase. This is a major departure from earlier work on reasoning-based information extraction: SOFIE used uniform weights except for entity disambiguation [30] and the work with Markov logic networks advocated setting weights by frequency analysis of the fact candidates (the "uncertain database") [15]. In PROSPERA, we associate the antecedent of a clause with a confidence weight about its constituent literals (elementary logical atoms). Specifically, for clauses of the form $occurs(p,e1,e2) \wedge expresses(p,R) \Rightarrow R(e1,e2)$, we use the confidence weight of the pattern p (see Section 4) as the weight of the entire clause. For clauses of the form $occurs(p,e1,e2) \wedge R(e1,e2) \Rightarrow expresses(p,e1,w2)$ we analogously use the confidence weight of the fact candidate $R(e1,e2)$ (see Section 4). The frequency of observing $p, e1, e2$ together (the $occurs$ predicate) is irrelevant, as this would unduly boost frequent observations regardless of their quality.

Our experiments show that the reasoner becomes much more robust by using the above weights, which are essentially dependent on seed facts (and the derived seed patterns). Clauses derived from functional dependencies, relational properties, or domain-specific consistency rules are given uniform weights.

6. DISTRIBUTED IMPLEMENTATION

To scale out our knowledge-harvesting system, we adopted the MapReduce programming model [13, 34] based on the abstractions of *mappers* and *reducers*. Mappers specify the computation that should be performed on each input record. Reducers specify how the output of the mappers should be aggregated to generate the final results. MapReduce computation is solely based on key-value pairs. Mappers work on input key-value pairs and generate intermediate key-value pairs. Reducers consume these intermediate pairs, with the same intermediate keys being passed to the same reducer. Reducers aggregate intermediate keys to emit output key-value pairs.

We have developed MapReduce algorithms for the three main phases of our architecture. We used the Hadoop open source implementation [34] and the HDFS distributed filesystem [27].

6.1 Pattern Gathering

Parsing documents for pattern gathering is trivially parallelizable as each document is scanned independently. No coordination is required between concurrent worker tasks. The input to the mappers are document identifiers (keys) and the corresponding document contents (values).

```
1. FUNCTION map(i, P_i)
2.     List N ← generateNgrams(P_i)
3.     FOR n_i ∈ N DO
4.         emit(n_i,1)

1. FUNCTION reduce(n_i, [v1, v2, v3, ...])
2.     support ← 0
3.     FOR v_i ∈ [v1, v2, v3, ...] DO
4.         support ← support + v_i
5.     IF support ≥ MINSUPPORT
6.         emit(n_i, support)
```

Figure 2: MapReduce pseudo-code for frequent n-gram-itemset mining

```
1. FUNCTION map(i, [e_1, p,e_2])
2.     IF isSeedPattern(p)
3.         FOR r ∈ R DO
4.             SeedOccurrence O ← [r, e_1,e_2]
5.             emit(p.id, O)

1. FUNCTION reduce(p.id, [O1, O2, O3, ...])
2.     List L ← { }
3.     FOR O ∈ [O1, O2, O3, ...] DO
4.         L.append(O)
5.     emit(p.id, L)
```

Figure 3: MapReduce pseudo-code for seed pattern confidence

The mapper performs checks on the sentences of the document, emitting triples of the form *(e_1, p, e_2)* for any pair of interesting entities e_1 and e_2. Additional processing of sentences, such as generating part-of-speech tags, is also performed in the mapper. Here the reducer merely serves the purpose of sorting and aggregating the emitted triples.

6.2 Pattern Analysis

The pattern analysis phase computes quality measures for seed patterns and uses these to generate fact candidates. The results of the pattern analysis phase are accomplished by a sequence of MapReduce algorithms; here we focus on the major tasks and how to distribute/parallelize them.

Generate N-gram-Itemset Patterns. The n-gram-itemset patterns are the primary representation on which pattern-similarity computation is based. Thus, the first task is to convert the previously collected raw patterns into this format. This entails identifying frequently co-occurring n-grams within the basic patterns via frequent-itemset mining [3]. To reduce the size of the input data to subsequent algorithms, we introduce a preprocessing step to perform dictionary encoding by replacing words and patterns with integer identifiers.

The pseudo-code for computing frequent itemsets is shown in Figure 2. The input to the mapper consists of the key-value pair of the pattern identifier (key) and the pattern itself (value). For each input pattern, mappers generate constituent n-grams and emit, for each n-gram, an intermediate key-value pair consisting of the (dictionary-compressed) n-gram as the key and a support of 1 as the value. The reducers gather support counts for any given n-gram and sum them up to obtain the final support counts. Only those n-grams whose support is above the specified values are emitted. Note that sequential algorithms for frequent-itemset mining are typically optimized to eliminate non-frequent itemsets as early as possible. When generating frequent itemsets of cardinality (or length in our case) k, the algorithm first prunes all infrequent (k-1)-grams. In contrast, our MapReduce algorithm greedily generates all itemsets and does batch pruning in the reducers. This is advantageous because 1) we are only interested in relatively short n-grams, typically 3-grams, and 2) the MapReduce paradigm is designed for batch processing and works best if coordination and communication across worker tasks is kept to a minimum.

Once we have the frequent n-gram itemsets, a second MapReduce algorithm (not shown here), is used to rewrite patterns into a form with frequent n-grams only, disregarding infrequent ones. This way we end up with n-gram-itemset patterns.

Compute Seed Pattern Confidence Values. Once all the patterns are in n-gram-itemset representation, we need to identify the seed patterns and compute their confidence values. To this end, we need to determine how often the pattern co-occurs with seed facts and how often it co-occurs with counter-seeds. We have developed two MapReduce algorithms for this purpose. The first one, shown in Figure 3, identifies seed patterns and tracks pattern-occurrence information. In each mapper, the (e_1, p, e_2) triples from the pattern-gathering phase are processed to test if p is a seed pattern. The mappers emit intermediate key-value pairs with the pattern identifier as the key and the seed occurrence as the value. The reducers combine all seed occurrences belonging to one pattern and emit all seed occurrences of every seed pattern. A second MapReduce algorithm (not shown here) uses this data to compute pattern confidence values.

Generate Fact Candidates. The large majority of the [e_1, p,e_2] triples from the pattern gathering have patterns p that do not precisely match any seed pattern. To identify new fact candidates and quantify their statistical evidence, we conceptually compute the similarity of p with all partially matching seed patterns q based on the Jaccard coefficient of the corresponding n-gram sets (see Section 4)

The easiest implementation would be an exhaustive algorithm where a mapper computes the similarity of a given pattern with *all* seed patterns and then emits the best (partial) match. However, this would have high computational costs because of many unnecessary comparisons. To accelerate the computation, we first build an inverted index on the n-grams of the seed patterns and use it to compute similarity scores more efficiently. For building the index, we follow standard MapReduce practice [13]. The optimized algorithm is shown in Figure 4.

The mappers consume non-seed patterns, with the pattern identifier as the key and the n-gram-itemset as the value. Each mapper first loads its relevant partition of the seed n-gram index and pattern confidence values into memory. Hadoop allows mappers to preserve state across different input pairs, therefore this information is loaded only once during the Hadoop job initialization. The mapper uses the index to compute matches between seed patterns and non-seed patterns. For each such match, the similarity score between the seed pattern and the non-seed pattern p_i is computed and added to a priority queue. The mapper then emits an output key-value pair consisting of the pattern identifier and the best matching seed pattern. This information is then passed down to the reasoner which makes the final decision on the goodness of the patterns during a given iteration.

6.3 Reasoning

To parallelize the MaxSat-based reasoning, the hypotheses about fact candidates and pattern goodness are formulated as a graph. Each fact candidate or pattern forms a vertex in the graph, and an edge is added between two vertices if they appear in a joint clause.

Relation	# Extractions			Precision			Precision@1000
	PROSPERA-6	NELL-6	NELL-66	PROSPERA-6	NELL-6	NELL-66	PROSPERA-6
AthletePlaysForTeam	14,685	29	456	82%	100%	100%	100%
CoachCoachesTeam	1,013	57	329	88%	100%	100%	n/a
TeamPlaysAgainstTeam	15,170	83	1,068	89%	96%	99%	100%
TeamWonTrophy	98	29	397	94%	88%	68%	n/a
AthletePlaysInLeague	3,920	2	641	94%	n/a	n/a	n/a
TeamPlaysInLeague	1,920	62	288	89%	n/a	n/a	n/a
AthleteWonTrophy	10	n/a	n/a	90%	n/a	n/a	n/a
CoachCoachesInLeague	676	n/a	n/a	99%	n/a	n/a	n/a
TeamMate	19,666	n/a	n/a	86%	n/a	n/a	100%

Table 1: Performance comparison between PROSPERA and NELL on sports relations

1. **FUNCTION** map(i, p_i)
2. $I \leftarrow$ loadSeedNgramIndex()
3. $C \leftarrow$ loadSeedPatternConfidenceValues()
4. PriorityQueue $Q \leftarrow$ { }
5. List $H \leftarrow$ computeHits(p, I, C)
6. **FOR** $h \in H$ **DO**
7. $h.similarity$ = computeSimilarity(p_i, $h.seedPattern$)
8. Q.insert(h, $h.similary$)
9. emit(Q.removeMin())

Figure 4: Mapper pseudo-code for pattern similarity and fact-candidate extraction.

Then the entire graph is partitioned into approximately equal parts, such that the number of edges that connect vertices in different partitions is minimized. Note that this approach may now disregard some constraints, but as MaxSat is an NP-hard problem our solution is approximate anyway. The fewer cross-partition edges are cut, the more constraints are preserved by the parallelized reasoning. Generating the graph is specified as a MapReduce job, the graph is then partitioned into *k* partitions, and the partitions are processed in parallel by reasoners on different compute nodes of the distributed platform.

The minimum-cut graph partitioning problem is also NP-complete. We employed a randomized, two-phase graph partitioning algorithm, based on methods by [18] and [19].

Phase one coarsens the graph into a smaller graph that is a good representation of the original graph. The coarsening reduces the size of graph by edge contraction until the graph is small enough. The basic technique is to pick an edge connecting vertices *u* and *v* and collapse the two vertices: a new vertex *w* replaces *u* and *v*. Edges previously linking *u* and *v* to other vertices are updated to point to the new vertex *w*. If both *u* and *v* have edges to another vertex *z*, then the weight of the edge from *w* to *z* is the sum of the weights of the two edges. This helps to ensure that a balanced partitioning of the smaller graph is also an approximately balanced partitioning in the original graph. In picking the edges to contract, we heuristically favor edges that contribute more to the overall min-cut objective function. These are the heavy edges in the coarsened graph. Initially, all edges have the same uniform weight, but as vertices are collapsed, some edges obtain higher weights. In each step, we randomly select a vertex and then choose its incident edge with the highest weight for contraction. This guards the edge from being cut in the second phase of the overall algorithm.

Phase two partitions the coarser graph and projects the resulting partitions back into the original graph. Once a coarsened graph with a specified maximum number of vertices is obtained, it is then directly partitioned into *k* partitions. We use graph-growing heuristics [19] to derive *k* partitions from the coarsened graph. For each *k*, we randomly select a vertex and grow a region around it until $|V|/k$ vertices are included, picking vertices that lead to smaller increase in the weight of the edges that are cut.

In our experiments, partitioning the graph this way did not notably affect the output quality of the Weighted-MaxSat solver, which is only an approximation algorithm anyway. The time for graph partitioning was short, usually 5 minutes at most for graphs with several 100,000's of vertices.

7. EXPERIMENTAL EVALUATION

7.1 Setup

Large-scale experiments were carried out on the English part of the ClueWeb09 corpus, which consists of 500 million English-language Web pages. Evaluations reported here were restricted to binary relations between entity pairs. We focused on two domains: 9 relations from the sports domain and 5 relations from the domain of academic relationships.

The sports domain was chosen in order to compare our PROSPERA approach to the results of the *NELL* (Never Ending Language Learning) experiment reported in [10], which had strong coverage of the sports relations and – very laudably – made all relevant data available on the ReadTheWeb site. To our knowledge, NELL is so far the largest and most ambitious experiment along these lines, with online data against which we could meaningfully compare our approach. We used the very same input as NELL: 10–15 seed facts and 5 counter-examples for each relation. In contrast to NELL, we did not have any human intervention during our runs. Also, NELL allowed 5 manually specified seed patterns as a-priori input, whereas PROSPERA used only seed facts and determined patterns autonomously.

The sports domain has no constraints other than type constraints; there are not even any functional dependencies. The academic domain, on the other hand, is an interesting choice as it has sophisticated constraints posing a stress-test to the reasoning phase in our PROSPERA system. For the academic relations, we used seeds obtained from the YAGO ontology. All instances for a given relation already in the ontology were treated as seeds. Counter-seeds were derived from instances of other relations in YAGO.

All experiments were performed on a Hadoop (0.20.1) cluster

with 10 server-class machines. Each machine has a Dual-Xenon E5530 2.4 GHz processor (16 physical cores), 48 GB RAM (DDR3), 1.5 TB iSCSI storage, and 1 Gbit Ethernet interconnect. The NELL experiment ran on the Yahoo! M45 supercomputing cluster, but no statements were given about the number of nodes used and their utilization.

Performance metrics of interest are *recall*, *precision*, and *runtimes*. Here recall refers to the number of extracted facts which are returned by each of the knowledge-harvesting systems, as there is no way of estimating the total number of truly correct facts that appear (in latent form with natural language) in the entire corpus. Precision was estimated by sampling the harvested facts and having a human judge assess their correctness. As our runs yielded much higher recall than NELL, we also ranked the resulting facts by their confidence and additionally determined the *precison@1000* for the 1000 highest-ranked facts of the largest relations. All precision assessments are based on 50 randomly sampled facts for each relation.

All data on the experiments reported here are made available as supplementary material on the Web site *www.mpi-inf.mpg.de/yago-naga/prospera/*.

7.2 Scalability Experiment (Sports Relations)

The scalability experiment aimed to evaluate the performance of our approach on large and noisy data for all three performance metrics. Table 1 shows the results of the experiment on sports relations. PROSPERA ran 6 iterations, and NELL ran 66 in total. We compare PROSPERA-6 (6 iterations) against NELL-6 (first 6 iterations) and NELL-66 (all 66 iterations). For the first four relations, both precision and recall numbers for NELL are given in [10] and its supplementary material. NELL was run on the other relations as well, but no recall/precision numbers were given.

PROSPERA has orders-of-magnitude higher numbers of extractions compared to NELL-6. Even NELL-66 still had substantially fewer results than PROSPERA. On the other hand, NELL had precision close to 100% for most of these relations, except for the *TeamWonTrophy*. For this relation NELL-6 has precision of 88% and further degradation can be seen in NELL-66 at 68%. Generally, [10] reported that NELL's precision would gradually go down with larger numbers of iterations. In contrast, PROSPERA's precision was consistently good and hardly varied across iterations. For the *TeamWonTrophy* relation, PROSPERA outperformed NELL on precision, for the other relations is was somewhat worse than NELL but still well above 80%. Note, however, that the precision across all extractions is misleading here, as PROSPERA returned a much higher number of facts. We also evaluated the *precison@1000* for the largest relations. Here, PROSPERA achieved 100%. So overall, our approach essentially achieved the same precision as NELL while giving much higher recall.

PROSPERA produces high-quality extractions in canonical form: relational facts between disambiguated entities. For example, across all extractions, all relational facts involving the team *Chicago Bulls*, always use the same consistent identifier for this team. This is different from the NELL output which has facts referring to names, returning, for example, both the *Bulls* and the *Chicago Bulls*. Our approach includes entity disambiguation, thus recognizing, based on the context of the name occurrence, if the string "Bulls" refers to the entity *Chicago_Bulls* or some other team with the ending "Bulls". Table 2 shows disambiguated output from PROSPERA in comparison to NELL extractions.

The total run-time of PROSPERA was about 2.5 days for the 6 iterations on the sports domain. This is in contrast to NELL's 6 or 66 days for the first 6 or all iterations, respectively. Note that NELL ran on a much larger cluster but also extract other relations and categories that we did not consider in our experiments. So the run-time numbers are not directly comparable. Nevertheless, especially in view of our much higher recall, the PROSPERA run-times look favorable.

In this experiment, the MaxSat-based reasoning constitutes only a small percentage of the total run-time. This is illustrated in Figure 5(a) where in every iteration, pattern analysis (the bottom part) took between 4 to 6 hours while reasoning took less than an hour. Not shown in Figure 5(a) is the run-time of the preprocessing for pattern gathering, which took approximately 20 hours. This time is included in the total figure of 2.5 days for gathering and 6 iterations of analyis and reasoning. Figure 5(b) shows that the number of extractions consistently increases over iterations; running more iterations would probably lead to further extractions.

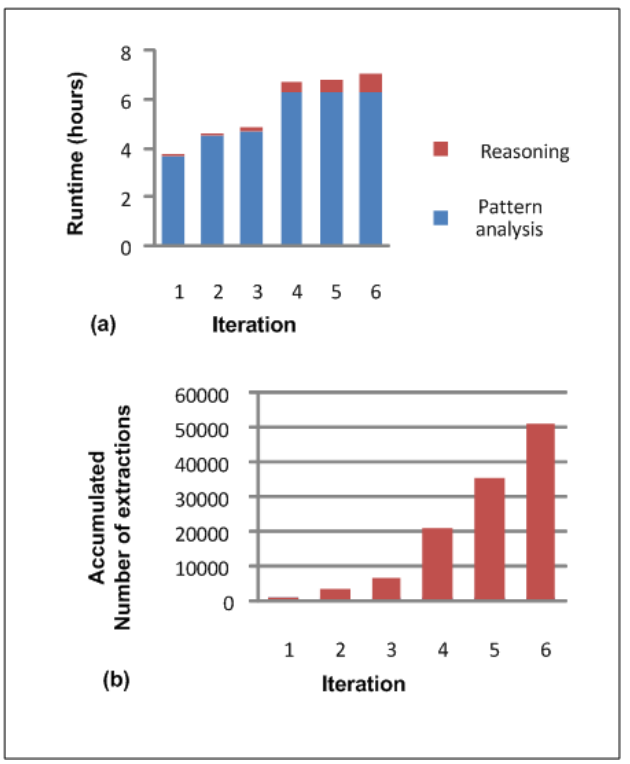

Figure 5: PROSPERA runtimes on the sports domain (a) and number of extractions across iterations (b).

To determine how individual components of our approach contribute to performance, we ran additional measurements with different variants of PROSPERA, selectively disabling one component at a time. For this study, we ran only 2 iterations. As the results in Table 3 show, reasoning plays a significant role in ensuring high precision. Without the reasoner, precision would be only 31%. Reasoning about the quality of the patterns ensures that patterns whose quality is uncertain do not lead to extractions. This initially leads to much lower recall, but for a good pattern, subsequent iterations gather enough support for the pattern. Thus, such a pattern is eventually accepted by later iterations of the reasoner, leading to increased recall over successive iterations. For example, in the first iteration, starting with the initial seed facts, the pattern *"would beat the"* only had a confidence weight of *0.5* for expressing the *TeamPlaysAgainstTeam* relation. By the fourth iteration, evidence in support of this pattern escalated and its confidence weight increased to *0.96*.

Relation	PROSPERA	NELL
AthletePlaysForTeam	(Ben_Gordon, Chicago_Bulls)	(Ben Gordon, Bulls)
TeamPlaysInLeague	(Chicago_Bulls, National_Basketball_Association)	(Chicago Bulls, NBA)
AthletePlaysForTeam	(Jason_Giambi, New_York_Yankees)	(Jason Giambi, Yankees)
TeamPlaysAgaintsTeam	(San_Francisco_Giants, New_York_Yankees)	(Giants, New York Yankees)
AthletePlaysForTeam	(Ben_Graham_(footballer), Arizona_Cardinals)	n/a
AthletePlaysForTeam	(Edgar_Gonzalez_(infielder), St._Louis_Cardinals)	n/a
TeamMate	(Metro_Prystai, Jim_Henry_(ice_hockey))	n/a

Table 2: Sample output from PROSPERA and NELL

Note that all this is fully automated in PROSPERA, whereas NELL had some manually designed seed patterns and used a small amount of human supervision after each iteration.

The confidence weights also play a vital role in the quality of our output, both for pruning bad patterns and for providing better statistical evidence to the reasoner about which candidates are more likely to be true. Without the use of weights, precision drops to 73%. On noisy Web data like the ClueWeb09 corpus, the new notion of n-gram-itemset patterns with confidence weights pays off well.

Method	# extractions	Precision	Runtime (hours)
Full-PROSPERA	3,174	90%	8.37
NoReasoner	157,610	31%	8.22
Unweighted	8,429	73%	8.32

Table 3: PROSPERA variants on sports relations (2 iterations).

To determine speedup obtained by parallelizing the reasoner, we reduced the number of reasoners by half and measured run-times for reasoning in the first iteration. Using half the reasoners, 5 in total (on 5 nodes of the cluster), took 7.6 minutes compared to 3.5 minutes when using 10 reasoners. This suggests a slightly superlinear speedup of 2.2, which is attributed to the fact that partitioning the candidate graph reduces the search space of the reasoner, resulting in faster execution times.

7.3 Constraints Experiment (Academic Relations)

As the sports-domain experiment did not have any advanced constraints, we carried out a second experiment that aimed to stress-test the constraint reasoning aspect of PROSPERA on five academic relations. We specified the following constraints in first-order logic:

- a student can have only one alma mater that she/he graduated from (with a doctoral degree);
- a student can have only one doctoral advisor (who had this role officially);
- the advisor of a student must have had a position at the university from which the student graduated;
- the advisor of a student must be older than her/his student.

We ran PROSPERA for two iterations only, using seed facts from the YAGO ontology; the results are shown in Table 5. Both the number of extractions and precision are high with the exception of the *hasAcademicAdvisor* relation which returned only few extractions with mediocre precision. Here, our approach of consistenly referring to canonical entities became unfavorable, as we could accept only facts where both student and advisor are known to YAGO (or Wikipedia, on which YAGO is based). The number of pairs that fulfilled this strict requirement, and also appeared in the ClueWeb09 corpus, simply was way too low.

In general, having rich constraints significantly improved precision as shown in Table 6. Without the reasoner, we obtained several birth dates for one person and in some cases more than five advisors for one person, a situation highly unlikely in reality. Without the reasoner precision dropped to 25%. Table 4 shows a few sample results for facts and their supporting patterns.

This experiment had longer run-times (ca. 18 hours for 2 iterations) due to the use of domain-specific constraints for the reasoner, and also because of the larger number of seed facts obtained from YAGO for the first iteration. The experiment again showed considerable speedup obtained from parallelizing the reasoner, with 5 reasoners (on 5 nodes of our cluster) taking 7.1 hours whereas 10 reasoners only took 2.7 hours.

Relation	# extractions	Precision	Precision @1000
bornOnDate	40,962	92%	97%
facultyAt	4,394	96%	98%
graduatedFrom	1,371	81%	n/a
hasAcademicAdvisor	46	75%	n/a
hasWonPrize	4,800	91%	100%

Table 5: Extracted facts and estimated precision for academic relations obtained from two iterations of PROSPERA.

Method	# extractions	Precision	Runtime (hours)
Full-PROSPERA	51,573	92%	18.3
NoReasoner	773,721	25%	13.1

Table 6: PROSPERA variants on academic relations

7.4 Discussion

The presented experiments are a proof of concept for the scalability of our approach. Each iteration of the analysis and reasoning phases takes only a few hours, with the Hadoop-based, parallelized PROSPERA system. In particular, even in the more demanding setting of the constraints-rich academic relations, the graph-partitioning-based distributed reasoner is fast enough to avoid bottlenecks.

In all experimental results, precision is very high and matches up against the high quality of the NELL results. In terms of recall, we achieved a much larger number of extracted facts, for some relations even orders-of-magnitude higher. Here the combination of richer patterns, statistically informative weights for clauses, and the resulting better input for the reasoner proved to be vital. As our studies with selectively disabling specific components of PROSPERA show, all building blocks are essential and their careful integration is key to the overall performance.

Relation	PROSPERA	Source pattern
facultyAt	(Richard_Axel, Columbia_University)	*'s group at*
graduatedFrom	(Albert_Einstein, University_of_Zurich)	*earned a doctorate from the*
hasAcademicAdvisor	(Miguel_Rolando_Covian, Bernardo_Houssay)	*student of*

Table 4: Sample output from PROSPERA

Regarding the quality of the output of the knowledge-harvesting systems, let us again emphasize that the facts by PROSPERA refer to canonical entities, whereas NELL's output refers to potentially ambiguous non-canonical names. When considering these approaches for further extending near-human-quality knowledge bases such as DBpedia, Cyc, or YAGO, this clean entity-level output is an important asset. In our experiments, the name-to-entity mapping heuristics worked very well. When sampling the accepted facts, we came across very few disambiguation errors, they had negligible influence on the overall precision.

8. CONCLUSION

This paper has addressed the ambitious goal of automatically constructing large knowledge bases from Web sources. It extended and improved prior work by projects like *KnowItAll*, *StatSnowball*, *ReadTheWeb*, and *YAGO-NAGA* in several ways. First, we introduced a new notion of n-gram-itemset patterns and associated confidence statistics. Second, we showed how to utilize this pattern statistics for MaxSat-based reasoning with informative clause weights, and we developed techniques for making the previously expensive reasoning much more efficient and parallelizable. Third, we integrated all building blocks into a Hadoop-based distributed system architecture for scalable knowledge harvesting that can achieve both high precision and much higher recall than prior methods. In large-scale experiments, we compared ourselves against the latest state-of-the-art competitor and demonstrated significant gains. Our experimental data is accessible on the Web site *www.mpi-inf.mpg.de/yago-naga/prospera/*.

Our ongoing and future work includes reaching out for more demanding relations that are expressed in very subtle ways but have sophisticated consistency constraints. For example, automatically distinguishing the relations *hasWonAward* and *nominatedForAward* is difficult (in domains like arts and sciences). In addition, we aim to systematically extract temporal scopes for all time-dependent facts [20, 31], for example, the timepoints of awards and the time periods for relational facts such as *worksFor* and *marriedTo*. Finally, we plan to integrate Web-extracted facts into the YAGO ontology, which has mainly relied on Wikipedia and WordNet, and releasing the extended knowledge base to the research community.

Acknowledgements

We are grateful to the European Union and to Google for supporting parts of this research, through the EU project Living Knowledge and a Google Research Award, respectively.

9. REFERENCES

[1] First International Workshop on Automatic Knowledge Base Construction (AKBC), Grenoble, France, 2010. *akbc.xrce.xerox.com*

[2] E. Agichtein, L. Gravano. Snowball: extracting relations from large plain-text collections. *ACM DL*, 2000.

[3] R. Agrawal, T. Imielinski, A.N. Swami. Mining Association Rules between Sets of Items in Large Databases. *SIGMOD*, 1993.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives. DBpedia: A nucleus for a web of open data. *ISWC*, 2007. *www.dbpedia.org*

[5] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, O. Etzioni. Open information extraction from the web. *IJCAI*, 2007. *www.cs.washington.edu/research/knowitall/*

[6] S. Brin. Extracting patterns and relations from the World Wide Web. *WebDB*, 1998.

[7] R. Bunescu, R. Mooney. Extracting relations from text: From word sequences to dependency paths. *Text Mining & Natural Language Processing*, 2007.

[8] M. J. Cafarella. Extracting and querying a comprehensive web database. *CIDR*, 2009.

[9] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka Jr., T. M. Mitchell. Coupled semi-supervised learning for information extraction. *WSDM*, 2010.

[10] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., T. M. Mitchell. Toward an Architecture for Never-Ending Language Learning. *AAAI*, 2010. *rtw.ml.cmu.edu/readtheweb.html*

[11] M.-W. Chang, L.-A. Ratinov, N. Rizzolo, D. Roth. Learning and inference with constraints. *AAAI*, 2008.

[12] P. Cimiano, J. Völker. Text2Onto – a framework for ontology learning and data-driven change discovery. *NLDB*, 2005.

[13] J. Dean, S. Ghemawat. MapReduce: a flexible data processing tool. *Commun. ACM* 53(1), 2010.

[14] A. Doan, L. Gravano, R. Ramakrishnan, S. Vaithyanathan. (Eds.). Special issue on information extraction. *SIGMOD Record*, 37(4), 2008.

[15] P. Domingos, D. Lowd. Markov Logic: An Interface Layer for Artificial Intelligence. *Morgan & Claypool*, 2009.

[16] O. Etzioni, M. J. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artif. Intell.*, 165(1), 2005.

[17] A. Jain, P. G. Ipeirotis, A. Doan, L. Gravano. Join optimization of information extraction output: Quality matters! *ICDE*, 2009.

[18] D.R. Karger, C. Stein. A New Approach to the Minimum Cut Problem. *J. ACM 43(4)*, 1996.

[19] G. Karypis, V. Kumar. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. *J. Parallel Distrib. Comput. 48(1)*, 1998.

[20] X Ling, D.S. Weld. Temporal Information Extraction. *AAAI*, 2010.

[21] N. Nakashole, M. Theobald, G. Weikum. Find your Advisor: Robust Knowledge Gathering from the Web. *WebDB Workshop*, 2010.

[22] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, S. Vaithyanathan. An algebraic approach to rule-based information extraction. *ICDE*, 2008.

[23] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 2006.

[24] S. Riedel, L. Yao, A. McCallum. Modeling Relations and their Mentions without Labeled Text. *ECML*, 2010.

[25] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3), 2008.A. Doan, J. F. Naughton, R. Ramakrishnan. Declarative information extraction using Datalog with embedded extraction predicates. *VLDB*, 2007.

[26] W. Shen, A. Doan, J.F. Naughton, R. Ramakrishnan. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. *VLDB*, 2007.

[27] K. Shvachko, H. Kuang, S. Radia, R. Chansler. The Hadoop Distributed File System. *MSST*, 2010.

[28] R. Srikant, R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. *EDBT*, 1996.

[29] F. M. Suchanek, G. Kasneci, G. Weikum. YAGO: a core of semantic knowledge. *WWW*, 2007. *www.mpi-inf.mpg.de/yago-naga/*

[30] F. M. Suchanek, M. Sozio, G. Weikum. SOFIE: a self-organizing framework for information extraction. *WWW*, 2009.

[31] Y. Wang, M. Zhu, L. Qu, M. Spaniol, G. Weikum. Timely YAGO: harvesting, querying, and visualizing temporal knowledge from Wikipedia. *EDBT*, 2010.

[32] G. Weikum, M. Theobald. From Information to Knowledge: Harvesting Entities and Relationships from Web Sources. *PODS*, 2010.

[33] M.L. Wick, A. Culotta, K. Rohanimanesh, A. McCallum. An Entity Based Model for Coreference Resolution. *SDM*, 2009.

[34] T. White. *Hadoop: The Definitive Guide.* O'Reilly, 2009.

[35] F. Wu, D. S. Weld. Automatically refining the Wikipedia infobox ontology. *WWW*, 2008.

[36] F. Xu, H. Uszkoreit, H. Li. A seed-driven bottom-up machine learning framework for extracting relations of various complexity. *ACL*. 2007.

[37] J. Zhu, Z. Nie, X. Liu, B. Zhang, J.-R. Wen. StatSnowball: a statistical approach to extracting entity relationships. *WWW*, 2009.