Applying Computational Complexity Theory and Cryptography to the Pursuit of Concept Understanding: *An Overview of Current Work*

Manuel Blum
(Grad Student: Ryan Williams)

1 Introduction

Our research is motivated by a simple but deep question: *can computers work with high-level ideas to solve problems as well as they can push symbols and crunch numbers?*

By "high-level", we mean a notion that is best illustrated with a common example. Compare almost every proof published in mathematics journals with a completely formal proof, having every detail down to the axioms; the first is what we call a "high-level" proof, and the second is the current kind of proof that a theorem-proving computer system generates. For another example, compare the best chess programs with the best chess Grandmasters. The best chess programs perform a highly optimized search over all possible moves, assigning numerical values to board positions and looking for moves that lead to the highest values. In stark contrast, Grandmasters only consider 2-3 moves at a time. Grandmasters are only looking at the most *relevant* and interesting moves; they think in terms of general, high-level trends in the game.

A "yes" answer to our question would suggest that perhaps computers can even achieve a kind of deep understanding in some areas of knowledge.

A "yes" answer to our research question would suggest that perhaps computers can even achieve a true understanding of some areas of knowledge, similar to that of humans. All of us constantly work with "high-level ideas" . We don't study the details of the road while driving, we merely keep track of general positions where other cars are, our approximate speed, *etc.* We intuitively understand what is important to our everyday tasks and what is not, but we lack a formal understanding of our own "understanding". A general mathematical theory of understanding would guide us in this direction, and possibly lead to more intelligent software. Under current software design, "true understanding" does not appear to be possible, and some philosophers such as Searle [Sea80] argue that computers with true understanding are an impossibility. While we do not assert that our work will answer the above question on a philosophical level, we believe that our research will yield significant progress on both a mathematical and practical level.

We approach the problem of equipping computers with high-level reasoning from a top-down, theoretical perspective. Rather than building up from examples or particular types of intelligent reasoning (*e.g.* the work of McCarthy, Lenat, and Reiter [McC90, LGPPS90, Rei80]), we have tried to step back as far away from the problem as possible, in order to grasp the larger picture. With such a broad perspective, we hope to discover underlying principles that pervade all high-level reasoning. A coherent and formal mathematical theory would bring new clarity and precision to a seemingly intractable subject.

Prima facie, the task of mathematically formalizing 'high-level ideas' may seem to be a circular proposal. That is, it seems that if a 'high-level idea' is made formal, then it would cease to be merely an idea! An idea is often inherently intuitive, and perhaps even vague. This is exactly the kind of object that a computer cannot manipulate. However, the indeterminacy of a process is not an impediment to its formalization. For example, in the definition of a nondeterministic algorithm, one defines a machines behavior via a relation between all potential "deterministic" behaviors of the machine. This is the sort of maneuver we take.

Our thoughts on high-level reasoning have led us to a simple but apparently powerful hypothesis. Roughly speaking, we believe that true understanding of a subject comes (as others *e.g.* Shank and Abelson [SA77] have observed) from the ability to effectively *summarize* information on the subject, and to distinguish what information is relevant to the subject from what is not. (Our "effectiveness" requirement is the entry point for computational complexity.) Starting from this hypothesis, we are working on a complexity-theoretic notion of *relevance*, initiating a study of the computational properties and features of relevant information under bounded resources. In general, the problem of relevant information is that the information that one truly wants to know is often much less than the information that one has (cf. [BL97] for a survey of relevance in machine learning). Our approach is distinguished by (a) its application of computational complexity and (b) its use of computational interaction, from theoretical cryptography.

We are beginning to discover interesting ways in which our relevance concept is connected to both current technological developments and modern theoretical computer science. In certain scenarios, relevant information takes the form of *summaries*, and in others, of *strategies*. Intuitively, a summary is a short and effective description of complex information. A strategy is a short and effective description of a problem-solving process. However, these "descriptions" are not like those normally studied in computer science: they can only be "decoded" by parties with sufficient background information (later, this will be made more formal). The effectiveness of such descriptions are highly dependent on the prior knowledge of the parties involved: a summary of world news has different meaning to a politician than a grade-schooler. This dependence shows itself rather strongly in the formalisms we are developing.

2 Summaries

We begin by observing a few key conditions needed for constructing good summaries: intent and background. A broad definition of summary should help us write computer programs that can intelligently summarize texts. Several automated summary programs exist, but they depend on natural language processing and thus have limited scope. The *intent* of such summaries is all the same: to condense a text to its major points. The *background* required to construct such a summary is similarly about the same: one just needs to be a competent English student. We would like a summary notion that is more language-independent, and more *robust* with respect to intent and background, so that objects other than just stories or articles can be summarized. For example, consider proofs of mathematical theorems: the *intent* of a proof summary is to help one easily reconstruct the full proof. The required *background* might be high-school math or graduate-school math, depending on content. This is quite different from summarizing news: one does not try to reconstruct the details of the news from a headline, and typically a generic reader type is assumed. A program that can automatically summarize with respect to many intents and backgrounds could have major applications in content delivery, Web search, theorem proving, and possibly other areas

like computer vision.

In the next two subsections, we compare and contrast an intuitive idea of summary to two other related topics: data compression and hashing. Later on, we will formalize our ideas more rigorously.

2.1 Summaries Versus Data Compression

We said above that a summary is a "short and effective description of complex data." From a computer science perspective, it would intuitively appear that the act of data summarization should be related to the act of data compression. Both tasks involve (1) a change in the way data is represented, and (2) the change results in a decrease in size.

However, in both cases, summarization and compression differ dramatically. In case (1), the change in representation of a summary is not as drastic as in compression. The "language" of a summary should be similar to the "language" of the original data. In contrast, all data compression schemes we know of either completely redefine the alphabet (*e.g.* Huffman codes) or augment the alphabet to such a degree that the representation of data is considerably altered (*e.g.* dictionary-based Lempel-Ziv compression). In fact, in principle we would say that a well-summarized text should be simple and clear to read, while a well-compressed text should have so many new alphabet symbols that it is *impossible* to read. (This should be clear to anyone who has tried to read the ZIP of a compressed ASCII file.)

In case (2), the primary intent of data compression is to obtain the smallest possible representation of the data. Sometimes a loss of content is acceptable (images) and in other cases it is not (executable files). However, data summaries often result in a significant loss of data. The intent of summarization can vary widely, and in general it depends on what the reader wishes to ascertain from the document in question, and the reader's background. A summarization of a theorem's proof would in principle outline each major step taken along the way, and hint at how the steps are connected to each other. This is primarily because a researcher reading the summary wants to know what mathematical ideas are used, and/or how he or she might reproduce a correct argument.

Compare this with the different kinds of information one might want from an article about the president (the Republican slant versus the Democratic one). For a less touchy (and more illustrative) example, suppose an avid reader of sports news wants a summary of the Pittsburgh versus New England game. Certainly the reader wants the score, and any decent summary should contain that. Beyond that, however, the content of a good summary varies greatly depending on the reader.

If the reader is a "Big Ben" Rothelisberger fan, he/she would care primarily about how Mr. Rothelisberger performed, in particular. A summary would need to contain this, to be relevant to the reader. A Tom Brady fan would probably care less about that, and would want more about Brady's performance. A fantasy football fan would want to know primarily how the players on his fantasy team did. (Such players could be on both teams.) A businessperson with substantive investment in one of the two teams would want to know who was injured on their invested team, if anyone– and a Cleveland Browns fan would just want to know what Pittsburgh players were injured.

So, unlike data compression, where one wishes to eliminate redundancy, we might say that in data summarization, one wishes to eliminate *irrelevance*. The casual mathematician does not want to know the proof details, just a few sentences about how the ideas come together. Removing irrelevance does not involve a change of alphabet, but it may involve a red marker. It may also

involve simplifying data, an interesting problem that, while enormously important, not very clear to us at the moment.

On the above, we could say that a summary of a document is a condensed version, written in similar language to the original, containing just the information that is relevant to a prospective reader. In this sense a summary is like the answer to a reader's unasked query: "what happens in this document that I should care about?"

2.2 Summaries Versus Hashing Techniques

Note that some formal definitions of ' summary" have been proposed (*e.g.* shingling), which involve using hash functions on parts of the data. In fact, some may argue that the hash value of a piece of data is a summary itself. However, hashing changes the representation of a document considerably. Now, in what sense might the hash value of a document be a good summary? Typically a hash value is not human-readable, but this is perhaps a minor point: each value in the range of the hash function could be replaced with a placeholder English sentence. The real question is, would such a summary convey the right information? Would it be *relevant*?

Suppose a reader has a document D, and would like to know if there are any other documents from a collection that are exactly like D. This problem is typically solved by choosing a small set of random hash functions (from a universal family), and comparing the hash values of D on these functions to the other documents' hash values. We would like to argue that if the reader's intent is to determine document equality, then this hashing scheme gives good summaries for that reader with high probability. However, in this case, every symbol of D is relevant to the task (nothing is irrelevant).

On the one hand, it looks in this setting that *understanding* of D is pretty divorced from being able to summarize D. On the other hand, if one just wanted to "understand" how D relates (in an equality sense) to other documents, then the above actually does qualify as a good way to summarize. Therefore, the *intent* or *purpose* of a summary is paramount to its success. This point will come out cleanly in our later formalization.

3 Summaries and Search Engines

We now discuss a hypothesized connection between search engines and automated summarizers. We intend to thoroughly explore this connection, finding empirical and mathematical evidence validating (or refuting) it. The problems of sorting and searching are among the oldest in computer science, and today they are more important than ever. Thanks to the Web, we currently swim in an ocean of information that needs vastly better organization. Web search has come a long way in helping us organize the billions of existing pages [BP98, KPR04], but substantial problems still remain. The current state of search on the Web is highly name-dependent: for many a person, if one knows the person's name, it is often not difficult to find information on him (if information is available); however if one knows only a description of the person that does not rely on special key words, it can be nearly impossible to find relevant information.

We would like to have a Web Search engine that could summarize pages in an intelligent way, so that only truly *relevant* results are returned. To this end, we are initiating a theory that aims to unify automated summarizing and search engines. A bird's-eye view of our thesis can be stated approximately as: *searching and summarizing are inverse processes.* That is, we contend that a

search engine which always produces relevant results is the *inverse* of a summarization function that, given a collection of documents, always produces a good short summary of the documents. Moreover, good search engines can help improve the quality of summarizers, and vice-versa.

While our thesis looks compelling, it is by no means self-evident. One supporting example to keep in mind is that of the "ideal" search engine: a human. Consider a professional movie critic. You forgot the name of a movie and you are sure the critic has seen it. Telling the critic a summary of that movie is an excellent "search query" to the professional movie critic. In the following, we describe the thesis in more detail, and present a few arguments for why it should be true.

3.1 The Inverse Search Problem

In the traditional search setting, the user enters a simple query and a collection of pages are returned. Here we consider a problem which is essentially the *inverse* of this. We believe this inverse problem may be at least as important as the traditional one.

What if the user would like a summary of a given page, or a list of keywords that appear in it? Suppose we had an extremely clever search engine that can take our English questions and return a plethora of highly relevant and interesting results.

For a set Y, let Y^k denote the collection of ordered k-tuples of elements in Y. The external behavior of a search engine is completely described by a function

$$S : \Sigma^n \to (\Sigma^m)^k,$$

where Σ is a finite alphabet, and $n << m$, $k << m$. Y represents the "pages" (in general, possible results) indexed by the search engine. The k represents the top-k results returned by the engine. So S takes a short string (the query), and returns a k-tuple of long strings.

What would it mean to compute S^{-1}? For $y \in Y^k$, let $y[i]$ be the ith component of y.

Let T be an i-subset of Σ^m, where $i \in \{1, ..., k\}$. We define

$$S^{-1}(T) \quad := \quad \text{the minimum length } x \text{ such that } T \in \bigcup_{i=1}^{k} S(x)[i].$$

Intuitively, the definition says that when S^{-1} is given at most k documents T, then S^{-1} returns the shortest search query for which T appears as the top-k results, in some order.

The primary thesis of our approach is the following, informally stated.

If for all queries x, $S(x)$ returns highly relevant search results for x,
Then for all T, $S^{-1}(T)$ is an effective *summarizer* of T.

One can imagine taking any small collection of documents, feeding it to S^{-1}, and obtaining a nice summary (or, given the way that Search is currently done, a nice list of KEYWORDS) for the set of documents. Effectively, what is returned is precisely the minimum amount of information one would need to tell S in order to get this set of documents to be the top search results. For our argument to work, we would need this search engine to contain *many* strings, enough that the minimum query contains the relevant features of this collection of documents that distinguishes it from other collections.

Currently, as search engines work, a query that puts a certain collection of documents at the top of search results would be some distinguishing keywords among those documents, as the function of a search engine is to distinguish the kind of documents you want from the documents you do not want, and this is currently the best way known how to do that. These keywords have to be so distinguishing that they keep all other pages from being at the top as well. The problem of course is that "irrelevant" information might distinguish the documents from others. I put it in quotes, as the information may not be relevant if the purpose is to find out what makes this set of documents different from all the others.

It seems then, that the problem of search and the problem of summarization might really be two different ways of looking at the same problem. What kind of news article search would be such that its inverse is a good summary? Clearly, not one that works on keywords or otherwise distinguishing language. It seems that the news articles should be pre-summarized, in some way. Small groups of consecutive sentences get summarized, the set of new summaries gets summarized, and so forth.

To do this, the search engine has to imagine what it would be like if the small group of sentences was indexed into the engine: 'what would be the best query for this small group?' That becomes the summary for those sentences.

3.2 Potential Application: Automatic Text Summarization

We briefly outline how our above ideas could apply to the area of Automatic Text Summarization, where one is given a document and the task is to find a short subset of the document's sentences that adequately summarize the important points (Mani [Mani01]). Text summarization algorithms in the literature rely heavily on natural language processing techniques.

Typically, for the case of Search, Σ is taken to be the set of words. Suppose that we instead define Σ to be a set of *equivalence classes*, where each class is a variety of closely related sentences. Now consider a search engine S for documents over this new alphabet. Queries would be a short sequence of sentences, and a good search result would be a short list of highly relevant documents that contain sentences similar to those in the sequence.

What would S^{-1} do? It would take a small collection of texts, and output a small sequence of *sentences* for which the collection is a highly relevant response. In particular, this small sentence sequence effectively *selects* these documents from all other documents covered by the engine. In that sense, the sentence sequence is an effective summary of the texts.

3.3 How Search Engines and Summarizers Can Improve Each Other

By acquiring feedback from users, it is possible for a search engine to improve its ability to both search and to summarize. We sketch below how such a system might work.

Here, we assume $k = |\Sigma^m|$. In particular, the search engine S returns a permutation of the strings in Σ^m.

- When a query q is *successful* for a user:

  (More formally, this means that some page $p = S(q)[i]$ is a desired page for the user, where i is very small.)

  then the search engine could build up partial solutions to the Inverse Search Problem using these good queries. (In particular, one could insert a row $[p, q]$ into an existing two-column

table.)

- When a query q is *unsuccessful*,

  (More formally, this means that the first desired page $p = S(q)[i]$ is such that i is very large.)

  then the search engine could suggest a "better" query for p. In particular, it can suggest $S^{-1}(p)$.

  Two possible outcomes can happen from this:

  1. The user could agree that $S^{-1}(p)$ was a better query. In this case, the user has learned more about how to properly use the search engine.
  2. The user could disagree with $S^{-1}(p)$, and perhaps offer another possibility for what a good query should look like. This information would in turn improve the both the search engine's search and inverse search (summarizing) capability.

A weak form of this type of suggestion is already used by Google, but mostly in the context of spell-checking. (When one queries for "inofrmation", Google first asks: *Did you mean information?*.)

We believe that a search engine that engages in more involved *interaction* with the user would exhibit a better *understanding* of the information it has. Theoretical computer science has studied the power of proof systems where a prover and verifier can carry on conversations about the theorem that needs to be proved. Many of the results from this area are both extremely counter-intuitive and insightful, giving lessons on how to perform seemingly impossible tasks. For example, the fascinating concept of *zero-knowledge* shows how a prover can convince a verifier of the truth of a theorem, without giving away any bits of the proof!

4 Introduction to Strategy Systems

Another vein of research on our agenda is how the notion of relevance arises in mathematical proof systems. We are investigating proof systems that allow for the possibility of strategies in their proofs; we call these "strategy systems". The notion of summary here takes the form of "proof strategies" or "proof ideas": succinctly stated nuggets of information that mentally place the prospective prover much closer to a working proof of a theorem than she would be otherwise. Proof strategies are invoked constantly in mathematics, and in a sense are crucial to its very enterprise. Due to our background, we have been studying proof strategies in theoretical computer science.

The notion of a "proof strategy" is like that of a proof idea/hint, but it is a hint of a certain kind. A proof strategy refers to a general technique that can be used to solve a variety of similar problems, provided the right context is given. Proof strategies can come in many forms. They can be mnemonics, they can look just like real proofs but with "holes", they could be a reference in a book, they can be one word, *e.g.* "random". As such, it is a daunting task for one to provide a formal characterization of what a proof idea is, without potentially missing a good example of a proof idea.

Our concept of relevant information handles the job nicely. In our treatment, a proof strategy is defined with respect to a certain context: the theorem to be proved, the Prover proving the theorem, and the shared knowledge between a Teacher who gives the proof idea, and the Prover, who receives this idea and attempts to prove the theorem. We have initially thought of this shared

knowledge as a collection of known theorems and proofs of them. Then, a proof idea for a theorem will just be a short message sent from the Teacher to the Prover, such that when the Prover considers the candidate theorem, the message, and the context, the Prover can efficiently compute a proof for the candidate.

Therefore, we are thinking of a "proof strategy" as a piece of information that is short in length, but so *relevant* to the proof that the Prover and his context that the strategy suffices in determining a proof. In this sense, a proof idea (or strategy) is an effective summary of the proof.

Strategy systems are related to zero-knowledge proof systems, in the sense that the goal of a strategy system is opposite to that of a zero-knowledge proof system. Informally, a zero-knowledge proof system is a communication protocol between two parties, a Prover and Verifier, with a rather counterintuitive property: the Verifier learns that a certain Theorem X is true, and that the Prover can prove it, but the Verifier learns nothing about the proof of Theorem X itself. In a zero-knowledge proof system, a Prover wishes to transmit as much information as possible such that the Verifier is reasonably convinced that a theorem is true, but the Verifier gains no knowledge about the theorems proof. In stark contrast, a Prover in a strategy system wishes to transmit the minimum amount of information to the Verifier, such that the Verifier can recover the entire proof. However there is another important aspect of strategy systems that does not arise in interactive proof systems: we allow for the Verifier to have prior knowledge. We do not require a strategy to work at all, unless the Verifier already has some stored examples where that strategy is used.

4.1 Example: Solving Boolean Satisfiability

Boolean satisfiability (SAT) is extremely important in both computer science theory and practice. Besides its status as the canonical NP-complete problem, SAT solvers are used as subroutines in many real-world domains such as planning, model-checking, automated reasoning, hardware verification, and knowledge representation. Despite its NP-completeness, in practice, SAT instances can often be solved rather efficiently.

Let F be a satisfiable CNF (conjunctive normal form) formula on n variables. Define a *backdoor set* S to be a subset of the variables of F such that, once the variables in S are set 'correctly', the remaining instance can be solved in polynomial time by a certain algorithm called a *sub-solver*. Thus, a minimum length subset S plus an assignment to its variables is sufficient for a sub-solver to find a satisfying assignment of F.

The size of a minimum backdoor set depends on the sub-solver at hand: the better the sub-solver, the smaller the backdoor set can be. In practice, it has been empirically demonstrated that many real-world instances have small backdoor sets of variables with respect to the heuristics commonly used, and this is a strong justification for why these instances are solvable (Williams, Gomes, Selman [WGS03]).

5 Outline of a Formal Theory

Finally, we present some pieces of our formal theory so far, showing how the pieces relate to ideas described informally in the preceding text.

5.1 Formal Setup

The Model. We consider two parties, a "Suggester" and a "Prover". The two parties communicate with each other, sending messages back and forth. There is a database B of "examples", which is organized in some manner and may or may not be shared between the two. There is an instance X of a problem Π to be "solved". Π has the property that, given a solution to one of its instances, one can efficiently check if the solution works. (Formally, $\Pi \in \mathsf{NP}$.)

There is no *a priori* restriction on what messages the Suggester can send. On the other hand, the Prover is required to be an efficient computational device, so messages cannot be (exponentially) long. This efficiency requirement is captured by putting a bound on how many steps it must take in order to compute a response to a Suggester's message, *e.g.* the Prover is restricted to run in polynomial time.

The Goal: The goal for the Suggester is to send a minimal length message M to the Prover, whereby with M, X, and database B, the Prover can efficiently reconstruct a solution to X.

M is defined to be a *summary of a solution to* X.

Intuitively, a summary contains just enough relevant information for the Prover to solve instance X, given background B. X determines the *intent* of the summary: a solution to X is the requirement. B determines the *context* of the summary: this influences the interpretation of M, and the extent to which M can be short.

It is important to note that we are not ruling out the possibility for the message M to be a "normal" compression, *i.e.* a ZIP file. Namely, M can be *far shorter* than a compressed proof because of available side information: the background B and intent X. That is, the zipped version of a proof can be much larger than the zipped version of its strategy.

5.2 Applications/Instantiations of The Model

We now review some direct instantiations of the model: particular choices of the Prover and Suggester that show our model can represent important relevance-based problems.

1. **Theorem Proving.** In this context, the message M becomes a "proof strategy", a shortcut towards proving X. Given the right strategy (tailored for the database), the Prover can reconstruct a proof of X. In [BW06], we discuss how one instantiation of this idea for Boolean satisfiability, via the notion of a "backdoor set" of variables.

2. **Web Search.** Suggester (as user) sends a search query to Prover (as search engine), who has access to database B of pages. The Prover sends a short list of pages to the Suggester. Here, X serves as some common side information known to both the user and search engine, *e.g.* the location of the user, perhaps his/her search history, and so on.

3. **Computational Advertising.** Computer science theory is becoming increasingly tied to microeconomics, due to the wealth of nice problems in online commerce [KPR04, MSVV05]. One area in its infancy is computational advertising, *i.e.* automatically presenting the right online ads to the right consumers. Here, we examine the case where a particular seller wants to advertise to a particular buyer. The most important thing for the seller is to keep the advertisements interesting. If the buyer reads too many ads filled with information that is irrelevant to her, she may lose patience and the potential sale is lost.

We can model this problem by letting Suggester and Prover be the seller and buyer, respectively. B becomes a list of items sold, where each item $(X, Y) \in B$ is defined by a name X and a list Y of all the features of x. (Thus the instance X becomes a particular item to be advertised.) The Prover has an unknown but efficiently computable preference function $F : B \to V$, where V is a set of values (for simplicity, think of $V = \{good, bad\}$).

Finally, the message M by the Suggester is a short sublist of Y, that is sufficient for the buyer to efficiently decide if $F(X, Y) = good$ or $F(X, Y) = bad$.

6 In Summary

To gain a better understanding of how computers can understand concepts and ideas, we introduced a formal, high-level definition of 'summary' and the related notion of 'strategy'. We argued that our notion is different from hashing and compression schemes, and gave a possible connection between the ability to summarize and the ability to search. We also briefly discussed proof systems that use strategies to prove theorems, and informally contrasted strategy systems with zero-knowledge proof systems.

References

[BL97] A. Blum and P. Langley. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence* 97:245–272, 1997.

[BDMP91] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. *SIAM Journal on Computing* 20(4):1084–1118, 1991.

[BW06] M. Blum and R. Williams. Outline of a Computational Complexity Theoretic Notion of Relevance. Manuscript, 2006.
Available at `http://www.cs.cmu.edu/~mblum/`.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks* 30:107–117, 1998.

[MSVV05] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani. AdWords and Generalized On-line Matching. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 264–273, 2005.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing* 18(1):186–208, 1989.

[Kle99] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM* 46(5):604–632, 1999.

[KPR04] J. M. Kleinberg, C. H. Papadimitriou, P. Raghavan. Segmentation problems. *Journal of the ACM* 51(2):263–280, 2004.

[LGPPS90] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. CYC: Toward Programs With Common Sense. *Communications of the ACM* 33(8):30–49, 1990.

[Mani01] I. Mani. *Automatic Summarization*. John Benjamins, Amsterdam, 2001.

[McC90] J. McCarthy. *Formalizing Common Sense: Papers by John McCarthy*. Ablex Publishing Corporation, Norwood, NJ, 1990.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence* 13:81–132, 1980.

[Sea80] J. R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences* 3(3):417–457, 1980.

[SA77] R. Shank and R. Abelson. Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures. Lawrence Erlbaum Associates, 1977.

[WGS03] R. Williams, C. Gomes, and B. Selman. Backdoors to Typical Case Complexity. *Proceedings of International Joint Conference on Artificial Intelligence*, pp.1173–1178, 2003.