Exploring an Information-Based Approach to Computation and Computational Complexity

D. E. Stevenson

442 R. C. Edwards Hall Department of Computer Science

Clemson University

PO Box 340974

Clemson, SC 29634-0974

April 1, 2002

Abstract

We present the background and justification for a new approach to studying computation and computational complexity. We focus on categories of problems and categories of solutions which provide the logical definition on which to base an algorithm. Computational capability is introduced via a formalization of computation termed a *model of computation*. The concept of algorithm is formalized using the methods of Traub, Wasilkowski and Woźniakowski, from which we can formalize the differences between deterministic, non-deterministic, and heuristic algorithms. Finally, we introduce a measure of complexity: the Hartley entropy measure. We provide many examples to amplify the concepts introduced.

1 Our Goals

> "That which you cannot explain, you do not understand."
>
> attributed to Albert Camus

We, as computer scientists, should be at least mildly concerned over the quality of software. As first documented by Charles Perrow [21] and so aptly continued by Peter Neuman in `comp.risks`, complexity is at least one of the culprits leading to failures. Perrow defines two axes for complexity: (1) the number of components and (2) the number of interactions

between and among components. Perrow contends that a large number of components and a large number of interconnections between them lead to "mind-boggling complexity." Dealing with such complexity requires at least qualitative, if not quantitative, reasoning to distinguish among various interactions.

In informal discourse, we sometimes say that one has "information overload" when one is overwhelmed with complexity. *Information* is one of those terms with several formal and informal meanings. First, a bit of history of the formal term.

The statistician R. A. Fisher seems to have originated the term *information*, which measured the signal in noisy data [**?**]. In the same year, 1926, Erwin Schrödinger used both the term and concept in physics [**?**]. **fisher26 and schrodinger26 references are not yet tracked down**. The term was first used in communications by R. V. L. Hartley in 1928 [14] then by Claude Shannon in 1948 [26]. Before and after Shannon there was much work in physics relating information to entropy. The initial connection was made by Leo Szilard in 1929 [27]. Leon Brillouin [3] made many contributions in 1962 with Shannon entropy. Roy Frieden [8] returns Fisher's original definition. Another use of the term *information* was made by Traub, Wasilkowski and Woźniakowski [28] who use the term to mean "values and structures available to describe a problem."

Why should we develop these ideas? Discussions with researchers in computational complexity lead to a realization that the Hamiltonian Cycle Problem has an information related difficulty: There seems to be no way to gain enough "information" about the solution so we can avoid having to look at all possible solutions. Our long-term goal is to develop a framework in which to understand formally what this statement implies.

This is a foundational paper: it seeks to explain complexity in terms of the amount of information that must be gained during an algorithm. Therefore, we propose a new understanding of old ideas rather than to producing novel ideas in the old framework. Our purpose is to develop concepts for analyzing difficulties in obtaining computational solutions.

Section 2 formalizes the concept of problem and problem statement. Section 3 discusses a critical point in the development: logical models and models of computation. In Section 4, we introduce the representation-based concepts due to Traub, Wasilkowski and Woźniakowski. This forms the basis for understanding our information based approach. Section 5 completes the representation-based development. Using computation traces to show the cu-

mulative gain of information is explained in Section 6. Section 7 gives several complete examples. We present conclusions in Section 9.

2 Problems, Problems, Problems

In considering the history of the development of science and mathematics, one is struck by the intertwining of problems and the development of methods. If one reads the history of mathematics (which is method oriented) without the sciences (which are problem oriented), one is inclined to having an overly romantic view of mathematics, its metaphysics, and its epistemology. A good book in the history of mathematics [2, 17, 18] helps dispel this notion; [18] is especially recommended. In particular, the relationship between problem and method is completely clouded in the mathematical classroom because the focus of mathematical knowledge is on method. A wonderful discussion of this point is made by Feynman in Chapter 2 of *The Nature of Physical Law* [7]. Unfortunately, any highly developed area seems to disconnect the seminal problems from the current subject matter. The first step leading to our framework is to focus on problems and their solution.

The approach outlined here centers on information, the link between information and entropy in thermodynamics, and the methodology of Traub, Wasilkowski and Woźniakowski [28], which they call *information-based*; unfortunately, the two concepts of information (entropy and TWW) are only distantly related. We shall refer to their methodology as *representation-based*. Information is a measure of complexity.

A *problem statement* is a pair of logical statements: the *givens* and a *to find* statement.

Example. Bin Packing From Garey and Johnson [9, p. 124][1]:

> "Consider the 'bin packing' problem: Given a finite set $U = \{u_1, u_2, \ldots, u_n\}$ of 'items' and a rational 'size' $s(u) \in [0, 1]$ for each item $u \in U$, find a partition of U into disjoint subsets $U_1, U_2, \ldots, U_k$ such that the sum of the sizes of the items in each U_i is no more than one and such that k is as small as possible."

□

Aside. I have specifically chosen this formulation to point out how much must be known by students being introduced to complexity issues. Indeed, our own expert understanding of the problem would be

[1] I specifically take the textual version and not the schematic version shown on page 226

more difficult without a metaphoric statement. In a cognitive sense, the understanding needed to properly formulate the problem is extra-mathematical. This, too, is a type of information. □

Aside. The style of the Garey and Johnson statement traces its history back to Euclid. The geometric researcher Menaechmus is given credit for the formulation of geometry as a series of problems. □

The givens and the solution (of the *to find*) statements are given in some logical system $\mathcal{L}$. Informally, a problem statement defines both structure and relationship. The structure is the definition of the objects in the problem and their relationship describes how objects are related to form answers.

Let the givens be stated as a relation[2] $\Delta(x)$ and the solution by a relation , (x, y). Any object x satisfying the givens Δ is called an *instance* of the problem. Any object y which satisfies , (x, y) is called an *answer*. Therefore, a particular problem statement might be denoted

$$p_{\mathcal{L}}[\Delta, ,]. \tag{1}$$

We call the pair $(\Delta, ,)$ the *constraints* of the problem. There is an implicit relation in the pair $(\Delta, ,)$; call this relation ρ:

$$\Delta \xrightarrow{\rho} , \tag{2}$$

ρ is the "answer" to the problem with no computation.

Informally, we would say that an algorithm ϕ solves the problem statement $p_{\mathcal{L}}$ if

$$\Delta(x) \wedge (\phi(x) = y \wedge), (x, y).$$

In the more usual computer science notation of Hoare triples, we write [5, 16]:

$$\mathbf{\Delta}(x)\{\phi\}, (x, y) \tag{3}$$

From this we say that one can *solve* a problem if one knows how to construct an algorithm which accepts instances and produces answers. Formally, let $\mathcal{P}$ be a category whose objects are problem statements and whose arrows are isomorphisms. Let Φ be a category whose objects are algorithms. The objects $\phi \in \Phi$ take problem instances as inputs $\Delta(x)$ and produces answers , (x, y). We seek functors to connect $\mathcal{P}$ to Φ

$$\mathcal{P} \xrightarrow{\Sigma} \Phi \ .$$

Σ is a *methodology* that maps problems to algorithms.

3 Solutions and Models

Our knowledge of a solution allows us to develop data structures and algorithms that take in an instance of the problem and construct answers. We know that there may be many data structures possible and for each data structure there could be many algorithms

[2] A relation is a statement in the formal language $\mathcal{L}$. We will assume the first order predicate calculus as our$\mathcal{L}$.

satisfying the problem. Any algorithm must satisfy the constraints in terms of computing capability that we call the model of computation. The purpose of this section is to define *model of computation*.

3.1 Models of Logical Theories and Models of Computation

In logic, it is not enough to simply lay out some arbitrary language and write something down that is syntactically well formed. We do not repeat the usual litany of definitions needed to pursue a thorough study of logic; see any good graduate text in logic such as van Dalen [29]. The principal component we need is the concept of a *model*. A model is a structure $\mathcal{M} = (A, F, R)$, where A is a set of objects, F is a set of functions, and R is a set of relations. To be a model, the logical language must be mapped to $\mathcal{M}$ in such a way that the evaluation of the axioms are satisfied by $\mathcal{M}$. That is, the model of the problem is anything that allows us to describe the instances and the answers.

3.2 Models of Computation

One implication of the work on the equivalence between various abstract computation devices seems to indicate that these abstractions form a category. Hennie [15] axiomatizes this class with respect to representation and capability; this axiomatization can easily be extended out of the number-theoretic realm: A *model of computation* or $\mathcal{MOC}$ is a structure that defines

1. Representations and constructors;
2. Rules of transformation;
3. Rules of sequence;
4. Rules of well-formed programs; and
5. Costs of storing representations and performing transformations.

We need such generality because programming and problem solving may take place outside the traditional computing; *e. g.,* quantum or DNA computing. Therefore, our ideas must be sufficiently rich to support objects and procedures other than computer programs.

Thus, models of computation are not synonymous with Turing machines or any other abstractions. Any real or abstract system imposes restrictions and costs. For example, the requirement to use object-oriented methods imposes real costs as well as perhaps eliminating some solutions.

A *program* is a statement conforming to the rules of well-formedness. A model of computation adds the concept of *costs*, which we must be able

to determine from the text of programs. An answer may be determined by many different algorithms and therefore have many different costs. The cost-oriented restrictions are called *criteria*. Criteria are used to optimize algorithms or to choose among several competing algorithms.

Summary. A problem is denoted by Eq. 1. The statement is made in a logical language. The problem can be faithfully represented[3] by many models. Different problems can be implemented by different models of computation. Therefore, we can think of the whole issue as

$$\Sigma : \mathcal{P} \times \mathcal{MOC} \rightarrow \Phi, \tag{4}$$

where $\mathcal{P}$ is a problem, $\mathcal{MOC}$ is a model of computation, and Φ is the class of algorithms that accept descriptions of instances of problems and produces answers. The exact form of Φ will differ based on $\mathcal{MOC}$.

Categorically, we have the following

$$(\Delta, ,) \xrightarrow{N_1} \mathcal{M} = (A, F, R) \xrightarrow{N_2} \mathcal{MOC} \tag{5}$$

In most realistic situations, $\mathcal{MOC}$ is taken as a given, leading to Equation 4 as the usual specification.

[3] This is not defined.

4 Information-Based Development

The functor $N = N_2 \circ N_1$ is called the *information map* in Traub, Wasilkowski and Woźniakowski [28]. As explained previously we call their method *representation-based* to distinguish their use of *information* and ours. Therefore, we call N the *representation functor*.

The representation functor plays a translation role that may be purely psychological. We now incorporate the approach in [28]. The computational problem $p_{\mathcal{L}}[\Delta, ,]$ is in the problem category defined in Section 2. It is convenient to think of the representation functors as providing two relations: the domain D and codomain C. The requirement is that the computational solution $\phi \in \Phi$ takes an element $d \in D$, called an *instance*, and produces an element $c \in C$, called an *answer*, so that the *logical statement* of the solution is satisfied.

If the the domain and codomain are both finite, then an information lossless coding should be possible and listing all the inputs and solutions might suffice. This is not always possible in the practical computing reality. Consider our bin packing example. The rational size of each object is positive, less than one. In this problem, the IEEE double precision

floating point acts as if there were a 54 bit mantissa and a ten bit exponent, giving 2^{64} possible values. We need 2^3 bytes of storage for one element and 2^2 bytes for the location of the value in the final solution. In other words, 2^{66n} bytes for an n element instance. This is of course not feasible since we are struggling to achieve terabyte (2^{45} bytes) storage.

Our points:

- Theoretical focus on a denumerable storage and denumerable computation steps does not help the finitist.
- How difficult it is to achieve an answer is probably related to how many elements we must contend with.
- This example represents a prototypical information based argument.

5 Information, Algorithms, and Error

Let the representation functor N map the domain Δ of the problem into a set D which serves as the domain for an algorithm ϕ. In the same vein, let M map the computed output the co-domain C.

5.1 Representation-based Approximation Concepts

Our approach is to focus on the development of the answer. Let ϕ be an algorithm that accepts instances and produces *sets* of answers. We allow for sets of answers to introduce nondeterminism. Following Traub, Wasilkowski and Woźniakowski [28], we introduce *representation-based* solutions:

$$\phi : D \times \mathbb{R}_+ \to 2^C$$

with properties

1. For all $d \in D, \phi(d, 0) \neq \emptyset$.
2. For all $\delta_1, \delta_2 \in \mathbb{R}_+$, if $\delta_1 \leq \delta_2$ then $\phi(d, \delta_1) \subseteq \phi(d, \delta_2)$.

We call $x \in \phi(d, \epsilon)$ an ϵ-approximation for the solution. In other words, ϕ induces a topology (see below).

Let ϵ be a non-negative real, then $\phi(d, \epsilon)$ is a subset of C. There is no reason to expect that $\phi(d, \epsilon)$ is unique because ϕ is nondeterministic. We form the equivalence class

$$V(d, \epsilon) = \{d_\epsilon : \phi(d_\epsilon, \epsilon) \cap \phi(d, \epsilon) \neq \emptyset\}.$$

If we define

$$A(d, \epsilon) = \bigcap_\epsilon \phi(d_\epsilon, \epsilon),$$

then we take A is the *answer*.

How *large* is A? The *radius of information* is its measure. Let

$$r(d) = inf\{\epsilon : A(d,\epsilon) \neq \emptyset, \forall d \in D\}$$

r is called the *local radius of information*. Let

$$\begin{aligned} R &= sup\{r(d) : \forall d \in D\} \\ &= inf\{\epsilon : A(d,\epsilon) \neq \emptyset, \forall d \in D\} \end{aligned}$$

is called the *global radius of information*. The most important thing to note is that, in general, the answers are not singletons.

Example. The theory in [28] is based on mathematical and numerical analysis. If the problem were "Find a root of a continuous real function which is in a given interval" then

- ϕ is some root finding algorithm.
- $d \in D$ is a function.
- $\epsilon \in \mathcal{R}_+$ is the tolerance.
- $\phi(d,\epsilon)$ is the interval around the root.
- $r(d)$ is the smallest safe tolerance, which is the tolerance that guarantees all the roots are captured..

□

Example. Heuristic solutions have many of the qualities of numerical routines. Suppose ϕ is a simulated annealing code to find the root of a system of equations. d is an instance of such a system. In this case, ϵ is the radius of the set containing the root. $r(d)$ is the smallest radius that can be used and still be guarantee a solution. □

5.2 Uniqueness, Non-Determinism, and Heuristics

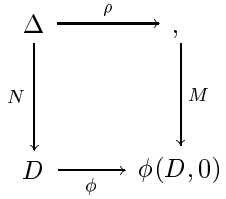
We can now offer the following interpretations:

- Problem $\mathcal{P}$ has a *unique* or *deterministic* solution if $\forall d \in D, \phi(d,0) = \{c\}$, where c is an answer.
- Problem $\mathcal{P}$ has a *non-deterministic* solution if $\forall d \in D, \phi(d,0)$ is not a singleton and $\forall c \in \{c\}$, $\phi(d,0)$ is an answer.
- Problem $\mathcal{P}$ has a *safe approximate* solution if $\phi(d,0)$ is non-deterministic.
- Problem $\mathcal{P}$ has an *unsafe approximate* solution if $\phi(d,0)$ is not a singleton and $\phi(d,0)$ has at least one element that is an answer and at least one element that is not an answer. This is the property we associate with heuristic solutions.

5.3 Error and Consistency

If either D or C has larger cardinality than Δ or , , respectively, there is an information loss. This naturally introduces the concept of *error*. The problem, however, is that there is no natural definition of error as there is in real analysis. However, the *consistency*

condition is given by the below diagram. We want the diagram to commute. This may be impossible for non-computable problems.



Error can be formalized by a series of definitions resembling the definitions of radii of information.

$$\begin{aligned} e(\phi, N, M, d) &= \inf\{\delta : \phi(N(d)) \in A(N, d, \delta)\}; \\ e(\phi, N, M) &= \sup_{d \in D}\{e(\phi, N, M, d)\} \\ &= \sup_{d \in D} \inf\{\delta : \phi(N(d)) \in A(N, d, \delta)\} \end{aligned}$$

where $e(\phi, N, M, d)$ is the local error and $e(\phi, N, M)$ is the global error of the algorithm.

Summary. We have developed a view of problems as stated in some logical system, discussed what the possible manifestations of the system might be (model), and introduced a the idea of a model of computation. We intend that programs will be written in the language of the model of computation but we must determine whether or not the things computed agree with the model. The process of determining whether or not these two sets are the same is called *validation*. To determine what is actually computed, we turn to denotational semantics.

6 The Trace of a Computation

The denotational semantics of programming languages [25, 13] has been studied since the 1970s with work by Scott in continuous lattices. An operational understanding of a language can be gained by considering the trace of a program. Consider the `fact`orial function:

```
fact(n) := if n=0 then 1 else n*fact(n-1).
```

One of the ways to understand the meaning of `fact` is through *unfolding* [1, 25], meaning successive substitution and simplification. For technical reasons [25, 13], the `fact` statement is encoded as a functional:

$F \stackrel{def}{=} \lambda fact.\lambda n.\ \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n * fact(n-1).$

Thus, the type of F is $\mathbb{Z} \to \mathbb{Z}) \to \mathbb{Z} \to \mathbb{Z}$. This is the motivation for Φ.

Let $graph(F_i)$ be all the values computed by F in i steps. Denote by $graph(F)$ what F should correctly compute. The condition we want F to meet is that it should compute what the relation ρ (Equation 2) specifies. Then

$$graph(F) = \bigcup_{i=0}^{\infty} graph(F_i)$$

and this defines a topology. Recall that a topology has the following properties [23]:

A topological space $\langle X, \mathcal{T} \rangle$ is a nonempty

set X of points with a family $\mathcal{T}$ of subsets (which we call open) possessing the following properties:

i. $X \in \mathcal{T}, \emptyset \in \mathcal{T}$;

ii. $O_1 \in \mathcal{T}$ and $O_2 \in \mathcal{T}$ imply $O_1 \cap O_2 \in \mathcal{T}$

iii. $O_\alpha \in \mathcal{T}$ implies $\bigcup_\alpha O_\alpha \in \mathcal{T}$.

Topological considerations have been important to theoretical computer science; see [6, 22].

Therefore, we have a mechanism for generating and checking sets of values. We need a mechanism to evaluate the *size of an open set.*

Example. Consider `fact(3)`. It takes four unfoldings to produce an answer. What do we know about `fact(27)` after four unfoldings? See [25]. □

The definitions of both the radius of information and the error also lead to a natural definition for topologies. The most important reason to emphasize the topological nature is that topologies are consistent, a requirement previously noted. This means that models have topologies [10]. It also turns out that once one has a topology, reasoning is possible. Therefore, one of our goals, how to reason about complexity, is realized.

7 Information in Science and Computer Science

Information allows us to tell that two things are different. This is a fundamental idea and at the heart of Brouwer's whole philosophy of intuitionism [30]. If there are 2^L elements in a set, then there are L bits of information required to discriminate among the elements. Indeed, Guiaşu comments [12, p. 29] "...information seems to represent a more primary step of knowledge than that of cognition of probabilities." Information is also used in the sense of "uncertainty". We are uncertain when there are multiple possible answers and no way to discriminate among them. Therefore, uncertainty and information are conjugate notions.

It is now time to introduce entropy and information as fundamental computational complexity measurements. The justification for using these is given immediately below: information is the ratio of entropies and entropy measures work versus uncertainty.

Warning concerning the term *negentropy.* Following Brillouin [3, Ch. 12], we define *negentropy* to be the negative of entropy so that a decrease in entropy is an increase of negentropy. We use *entropy* only for the thermodynamic concept. The

unfortuate use of entropy for the non-thermodynamic situation leads to confusion. The confusion is caused by Hartley and Shannon who defined *entropy* to be the negative of the value called entropy by physicists. For uniformity, we use *negentropy* throughout these sections dealing with calculations; *e. g.*, Shannon *negentropy*.

Why would we use a physical concept such as negentropy in computer science? One of several views of entropy (the physics term) in thermodynamics is "... the net increase in entropy that occurs during an irreversible process can be associated with a change in state from a less probable to a more probable state[31]" The thermodynamic relationship is stated as

$$T\Delta S = \Delta Q$$

where T is temperature, S is entropy and Q is heat energy. A perhaps better way to state this is $\Delta S \propto \Delta Q$; *i. e.*, the change in entropy is proportional to the change in energy. And "work" equates to "computation".

There are several interesting measures of negentropy. First, *Hartley negentropy* [14] defined as

$$H(A) = \log_2 |A| \tag{6}$$

where A is a set and $|A|$ is the number of elements in the set. The second is *Shannon Negentropy* which uses probabilities. Let p_i be the probability that the ith value is the desired value. Then Shannon negentropy [26, 11, 20] is defined by

$$H = -\Sigma p_i \log_2 p_i \tag{7}$$

Shannon negentropy is interpreted as the average information per symbol. As they have the same basic properties, we will have occasion to deal with both.

Shannon negentropy reduces to Hartley negentropy if the distribution is uniform.

$$\begin{aligned} H &= -\Sigma_{i=1}^{|A|} p_i \log_2 p_i \\ &= -\Sigma_{i=1}^{|A|} \frac{1}{|A|} \log_2 \frac{1}{|A|} \\ &= -\log_2 \frac{1}{|A|} \\ &= \log_2 |A|. \end{aligned}$$

7.1 Measuring Information and Its Change

The change in information (which is linked to thermodynamic negentropy as discussed above) is the change in negentropy. That is, the information gained by going from state S_k to S_{k+1} in the uniform case is

$$\begin{aligned} I &= H(S_k) - H(S_{k+1}) \\ &= \log_2 |A_k| - \log_2 |A_{k+1}| \\ &= \log_2 \frac{|A_k|}{|A_{k+1}|}. \end{aligned} \tag{8}$$

Prime Example: British Museum Algorithm. In the British Museum algorithm, the curator searches for something by always returning to the

starting point with an object. If the object is not the object sought, the search starts over. There are actually two possible interpretations which would be sampling *with* and sampling *without* replacement. From an information standpoint, *no* information is gained on the replacement case, since

$$H = -\Sigma_{i=1}^{n} p_i \log_2 p_i$$

is a constant. At each step in the *without* replacement there are fewer objects, hence H increases monotonically

$$H_n = -\log_2(n+1)$$

and the information gained on this transition is

$$I_n = -\log_2 \frac{n+1}{n}.$$

□

British Museum Problem, Revisited. To properly account for the searching the British Museum for *two* objects, the worst case time is the product $|A| \times |B|$. The (Hartley) negentropy of $|A| \times |B|$ is given by

$$H(AB) = \log_2 |AB| = \log_2 |A| + \log_2 |B|.$$

□

Principle 1 (Maximum Information Gradient) *A program should be designed so as to add the maximum amount of information to that which is already known. Put another way, take the step that leads to the greatest global reduction in uncertainty.*

7.2 An Example using Factorial

A simple example is in order. Consider once again `fact` as it is executed on a 32-bit machine.

```
fact(n) := if n=0 then 1 else n*fact(n+1).
```

What can we say about `fact(3)`? Think of this in terms of traces.

1. At the beginning, `fact(3)` could have any value in the interval $[0, 2^{31} - 1]$. Therefore, its initial negentropy is

   $$H = \log_2(2^{31}) = 31$$

2. Doing one step unfolding, we get `fact(3) = 3*fact(2)`. We have done some computation (work), so we should have gained some information. What is that information? We now can say that any answer must be divisible by 3. How many numbers are there? One-third of them. That means that the new space is $(2^{31} - 1)/3$. The third step removes every number not also divisible by 2. The negentropy at each step is

   $$\begin{aligned} H_0 &= 4.66 \times 10^{-10} \\ H_1 &= 1.44 \times 10^{-8} \\ H_2 &= 7.94 \times 10^{-8} \end{aligned}$$

   As predicted, the negentropy increases. There is no increase of negentropy if we go one step more

since the division by one does not contract the set. That is predicted by the fact that $H_2 = H_3$. Using the definition of information in Eq. 8 we get

Step	Number	Negentropy	Information
0	214783647	31.0	
			1.053
1	715827882	29.415	
			1.0
2	357913941	28.415	

At the end of the computation, we have a set which has numbers of the form

$$1 \cdot 2 \cdot 3, 1 \cdot 2 \cdot 3 \cdot n, \ldots.$$

The correct one must be $1 \cdot 2 \cdot 3$ since the other numbers require more information to differentiate them.

□

7.3 Algebra of Information

This is based on [3, 4].

We want to have as our primary concept of information a measure of how difficult it is to specify or find an object in the sense of constructing or computing the object. This interpretation comes originally from Salmonoff [24] and Kolmogoroff [19]. Therefore, exact information values are determined by the problem. Let us use $I(x)$ to identify the information in the object x.

If $B \subset A$, then the information difference is the I measure of the set difference.

$$I(A - B) = \log_2(|A| - |A \cap B|)$$

Let X and Y be sets. A relation on a set is a subset of the all the possible pairings,

$$R \subseteq X \times Y$$

with

$$\begin{aligned} R_X &= \{x|(x,y) \in R\} \\ R_Y &= \{y|(x,y) \in R\} \end{aligned}$$

Simple measures are $I(X) = \log_2 |R_X|$.

7.4 Joint Information and Information Under Hypothesis

Joint information for R is obvious since it is just a set. $I(X,Y) = I(R)$.

There is a concept of *contingent*. How much information do I have about R if I know something about the value of Y? This is called *conditional information*.

$$I(X|Y) = I(R|R_Y) = I(X,Y) - I(Y).$$

Conditional information measures the average imprecision regarding choices of X values for a given Y

value.

7.5 Total Information and Negentropy

Information transmission is a measure of the constraints between sets.

$$I(X) + I(Y) \geq I(X,Y) = I(X) + I(Y|X).$$

Negentropy is the average conflict between sets. In our simple case, negentropy is the average information content of the sets, which is

$$H(A) = \Sigma_{|A_i|} I(A_i)$$

Example. Let A be a list of positive integers that are used as a lookup table. What is the information worthiness for sorting a list to be used in searching?

1. Unsorted case. In the unsorted case, we have something similar to the sampling without replacement. The original number of possible elements in the first element of the table is $2^{31} - 1$. Because the table is unsorted, no conclusion concerning the other elements is possible by examining the first element. Therefore, the second set of possibilities is $2^{31} - 2$. Using Hartley negentropy, the information gained in the first step is $\log_2 2^{31} - 1/2^{31} - 2$, and this sequence proceeds, so in general the information gained at each step is $\log_2 i/(i-1)$.

2. Sorted case. In the sorted case, each step often results in large reductions of the set of remaining possibilities. As before, the number of possibilities in the first element is $2^{31} - 1$. Let A_1 be the value in the first location of the list. If the key sought is not A_1, then there are two possibilities:

   (a) If the key is less than that in A_1, then the key cannot be in the table and new set of possibilities is empty.

   (b) If the key is greater, then the set of possibilities is less than $2^{31} - 2$, it is$2^{31} - 1 - A_1$.

   In general, then, the information is given by $\log_2(2^{31} - 1 - A_{j-1})/(2^{31} - 1 - A_j)$.

□

Example. In the binary search *method*, we assume a sorted vector of values $u_i, u_i \leq u_{i+1}$. A target key v is, in principle, to be tested against each u_i. If there is one u_i that matches, then the search *succeed*s; otherwise it *fail*s. The method is

```
if( u_i=v ) then succeed
else if( u_i < v ) then ``go left''
else ``go right''
```

Before the first move, we know nothing. Assume the usual 32-bit integers; therefore, the negentropy is $H = 32$. If S_i is the set remaining after the ith test, then $S_0 =$ all values. S_i is still sorted

and has a midpoint of m_i. The first test gains a large amount of information, in general, because it "halves" the set S_0. Further tests do not make large strides until the other bound is found. This move may never be taken.

The principal of maximum information gradient is violated since we do not design the algorithm to determine the bounds, largely because such bounds are not *useful* information.

Space precludes our exploring this well discussed area, due to Brillouin, known as *bound* versus *free* information. □

8 Information Independence

Let $\mathcal{L}$ be a logical system and $\mathcal{A}$ be a programming language (in its most general sense). Let $\mathcal{R}$ be a relational structure for both $\mathcal{L}$ and $\mathcal{A}$. A program S in $\mathcal{A}$ is a procedures that maps states to states. Let Σ be the set of states for a problem $\mathcal{P}$. The *meaning relation* for S with interpretation $\mathcal{I}$ is a relation

$$M_{\mathcal{I}}S : \Sigma \rightsquigarrow Sigma \tag{9}$$

such that

$$M_{\mathcal{I}}S\sigma = \begin{cases} \sigma' & \text{if } S \text{ terminates} \\ \perp & \text{otherwise} \end{cases} \tag{10}$$

A *configuration* in one argument set to a procedure and an interpretation. A *configuration space* $\mathcal{C}$ is the set of all configurations.

Rationale. A construct in a program has an evironment. A configuration is one environment for a construct. We think of the configuration space as a behavior table.

Definition 1 *A configuration A is* informationally independent *of a configuration B with respect to a predicate ϕ if and only if knowlege of the status of A with respect to a property ϕ does not change the knowledge of the status of B with respect to ϕ.*

Rationale. The initial motivation for this definition is to capture the notion of determining the circumstances under which all possible inputs tested. The terminology is chosen to be similar to the probabilistic concept of independence.

Theorem 1 *If, for all A and B in a configuration space $\mathcal{C}$, A is independent of B with respect to ϕ, then to determine the configurations that are true for ϕ, the algorithm must be run against all elements of $\mathcal{C}$.*

9 Conclusion

I have introduced an outline of a theory of computing that focuses on the information. I have related these ideas to a wide range of concepts in computer science. We have been able to reproduce standard arguments with a bit more precision. This is the first

requirement of any new set of ideas: they must at least explain what we know.

The framework:

1. A problem $\mathcal{P}$ is a category whose objects are *problem statements*. Problem statements are logical statements that set the constraints $(\Delta, ,)$, where Δ is the instance relation and , is the solution relation.

2. An *algorithm* $\phi \in \Phi$ solves a problem $\mathcal{P}$ if it fulfills the Hoare relation $\Delta\{\phi\}, .$

3. A problem is a logical statement of constraints but an algorithm is based on a *model of computation* $\mathcal{MOC}$. The $\mathcal{MOC}$ sets the costs and the language of the algorithm. *Criteria* of choice are used to choose algorithms based on costs.

4. Algorithms actually compute a series of *approximations*. This introduces *radius of information, error* and *consistency*. This specifies the topology of the solution.

5. Traces generate a topology of solution by a particular algorithm. The generated topology defines the nesting of sets; *i. e.*, the convergence.

6. Algorithms can be designed following the *principle of maximum information gradient*: take the step that guarantees the largest decrease in negentropy.

The theory is far from being formalized and certainly does not yet answer the question laid out in Section 1: is there enough information in an instance of an NP-complete problem to ever be able to develop a polynomial solution?

References

[1] H. P. Barendregt. *Lambda Calculus: Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North Holland, 1981.

[2] Carl B. Boyer. *A History of Mathematics*. Wiley, 1989.

[3] Leon Brillouin. *Science and Information Theory*. Academic Press, 1962.

[4] Gregory J. Chaitin. Algorithmic information theory. *IBM J. of R & D*, pages 350–359,496, 1977. Reprinted in *Information, Randomness, and Incompleteness*.

[5] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1989.

[6] H. Ehrig. *Categorical Methods in Computer Science: With Aspects from Topology*. Springer Verlag, 1989.

[7] Richard P. Feynman. *The Character of Natural Law*. MIT Press, 1965. QC28.F5 1968.

[8] B. Roy Frieden. *Physics from Fisher Information: A Unification*. Cambridge University Press, 1999.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Incompleteness*. W. H. Freeman, 1979.

[10] R. Goldblatt. *Topoi: The Categorial Analysis of Logic*. North-Holland, 1984.

[11] Soloman Goldman. *Information Theory*. Prentice-Hall, 1953.

[12] Silviu Guiaşu. *Information Theory with Applications*. McGraw-Hill, 1977.

[13] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[14] R. V. L. Hartley. Transmission of information. *Bell Sys. Tech. J.*, pages 535–563, 1928.

[15] Fred Hennie. *Introduction to Computability*. Addison-Wesley, 1977.

[16] C. A. R. Hoare and P. E. Lauer. Consistent and complementary formal theories of semantics in programming languages. *Acta Informatica*, 3:135–153, 1974.

[17] Morris Kline. *Mathematical Thought from Ancient to Modern Times*. Oxford University Press, 1972.

[18] Morris Kline. *Mathematics: The loss of certainty*. Oxford University Press, 1980.

[19] A. N. Kolmogoroff. Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, IT-14:662–664, 1968.

[20] Soloman Kullback. *Information and Statistics*. Wiley, 1959.

[21] Charles Perrow. *Normal Accidents*. Basic Books, 1984.

[22] G. M. Reed, A. W. Roscoe, and R. F. Wachter. *Topology and Category Theory in Computer Science*. Oxford Science Publications, 1991.

[23] H. L. Royden. *Real Analysis*. MacMillan, 2 edition, 1968.

[24] R. J. Salmonoff. A formal theory of inductive inference. *Information and Control*, 7(2,3):1–22,224–254, 1964.

[25] D. A. Schmidt. *Denotational Semantics*. Allyn and Bacon, 1986.

[26] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. U. of Illinois Press, Champaign, IL, 1949.

[27] Leo Szilard. On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings. In John. Archibald Wheeler and Wojciech Hubert Zurek, editors, *Quantum Theory and Measurement*, chapter Irreversibility and Quantum Theory, pages 539–548. Princeton University Press, 1983.

[28] J. F. Traub, G. W. Wasilkowski, and H. Woźniakowski. *Information, Uncertainty, Complexity*. Addison-Wesley, 1983.

[29] D. van Dalen. *Logic and structure*. Springer-Verlag, 1994.

[30] Walter P. van Stigt. *Brouwer's Intuitionism*. Elsevier Science Publishing Company, Inc., 1990. QA29.B697S25.

[31] Gordon J. Van Wylen and Richard E. Sontag. *Fundamentals of Classical Thermodynamics*. John Wiley & Sons, 1985.