Proceedings

Application of Information Theory Entropy as a Cost Measure in the Automatic Problem Solving[†]

Eugene Eberbach[1,*]

[1] Dept. of Computer Science and Robotics Eng. Program, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609-2280, USA; eeberbach@wpi.edu

* Correspondence: eeberbach@wpi.edu ; Tel.: +1-508-831-4937

† Presented at the IS4SI 2017 Summit DIGITALISATION FOR A SUSTAINABLE SOCIETY, Gothenburg, Sweden, 12-16 June 2017.

Published data: 9 June 2017

Abstract: We study the relation between Information Theory and Automatic Problem Solving to demonstrate that the Entropy measure can be used as a special case of $-Calculus Cost Functions measure. We hypothesize that Kolmogorov Complexity (Algorithmic Entropy) can be useful to standardize $-Calculus Search (Algorithm) Cost Function.

Keywords: Information Theory; Entropy; superTuring Models of Computation; Automatic Problem Solving; $-Calculus; Machine Learning, ID3

1. Introduction

Information Theory Field founded by Claude Shannon in 1948 [7] is a branch of statistics that is essentially about uncertainty in communication. Shannon showed that uncertainty can be quantified, linking physical entropy to messages and defined the **entropy** of a discrete random variable X as $Entropy(X) = -\sum_{i=1}^{n} P(x_i)\log_2 P(x_i)$.

A (key) result of Shannon entropy is that $-\log_2 P(x_i)$ gives the length in bits of the optimal prefix code (e.g., Huffman code) for a message x_i. Similar like for probabilities, the conditional entropy has been defined to express mutual information, and entropy for continuous random variables.

It looks that Information Theory can be applied practically to anything, including coding theory, communication, data mining, machine learning, physics, bioinformatics. In this paper, we investigate the relations between Information Theory and Automatic Problem Solving.

Universal Problem Solving Methods are the part of AI and Theoretical Computer Science [2-4,6]. However, automatic problem solving requires construction of the universal algorithm, and this is a Turing machine unsolvable problem. The importance of automatic problem solving methods is so tremendous that even partial solutions are very desirable:

- The never ending dream of universal problem solving methods resurrect throughout all computer science history:
  - Universal Turing Machine (UTM, Turing, 1936)
  - General Problem Solver (GPS, Simon and Newell, 1961)

  - Evolutionary Algorithms (Holland, Rechenberg, Fogel, 1960s)
  - Prolog and Fifth Generation Computers Project (1970/80s)
  - Genetic Programming Problem Solver (GPPS, Koza, 1990s)
  - $-Calculus ([2-4])
- Negative Results: *a universal algorithm does not exist* - halting/decision problem of UTM (Turing, 1936) can be represented as a special instance of the universal algorithm, No Free Lunch Theorem (Wolpert, Macready, 1997)
- Positive Result: *a universal algorithm can be indefinitely asymptotically approximated* (Eberbach, CEC'2003).

For such approximation, the models of computation more expressive than TM are needed. They are called *superTuring* or *hypercomputational models of computation* or *superrecursive algorithms* [1,3].

2. Automatic Problem Solving: $-Calculus SuperTuring Model of Computation

- $-Calculus (read: Cost Calculus, [2-4] is a process algebra using anytime algorithms [6] for interactive problem solving targeting *intractable* and *undecidable* problems. It is formalization of resource-bounded computation (anytime algorithms) [6] guaranteeing to produce better results if more resources (e.g., time, memory) become available. Its unique feature is support for problem solving by incrementally searching for solutions and using cost performance measure to direct its search.

We can write in short that $-Calculus = Process Algebra + Anytime Algorithms

- Historically $-Calculus has been inspired both by λ-Calculus and π-Calculus:
  - Church's λ-Calculus (1936): sequential algorithms, equivalent to TM model, built around *function* as a basic primitive.
  - Milner's π-Calculus (1992): parallel algorithms, a superTuring model, but no support for automatic problem solving, built around *interaction.*
  - Eberbach's $-Calculus (1997): parallel algorithms, a superTuring model, built-in support for automatic problem solving (kΩ-optimization), built around *cost.*
- The kΩ-optimization meta-search represents this "impossible" to construct but "possible to approximate indefinitely" universal algorithm, i.e., it approximates the universal algorithm. It is a very general search method, allowing to simulate many other search algorithms, including A*, minimax, dynamic programming, tabu search, evolutionary algorithms.

$-Calculus Syntax:

- *Simplicity, everything is $-expression,* open system, prefix notation with potentially *infinite* number of arguments, and consisting of simple and complex $-expressions
  - Simple (atomic) $-expressions
    - send (→ a P1 P2 …) send Pi through channel a
    - receive (← a X1 X2 …) receive Xi from channel a
    - cost ($ P1 P2 …) compute cost of Pi
    - suppression (' P1 P2 …) suppress evaluation of Pi
    - atomic call (a P1 P2 …) and its definition (:= (a X1 X2 …) ^P^)
    - negation (¬a P1 P2 …) negation of atomic function call

  - Complex $-expressions
    - general choice (+ P1 P2 …) pick up one of Pi
    - cost choice (? P1 P2 …) pick up Pi with the smallest cost
    - adversary choice (# P1 P2 …) pick up Pi with the highest cost
    - sequential composition (. P1 P2 …) do P1 P2 ... sequentially
    - parallel composition (| | P1 P2 …) do P1 P2 … in parallel
    - function call (f P1 P2 …) and its definition (:= (f X1 X2 …) P)

$-Calculus Operational Cost Semantics:

- Search methods (and kΩ optimization, in particular) can be
  - *complete* if no solutions are omitted,
  - *optimal* if the highest quality solution is found,
  - *totally optimal* if the highest quality solution with minimal search cost is found.
- Search can involve single or multiple agents; for multiple agents it can be
  - cooperative ($-calculus cost choice used)
  - competitive ($-calculus adversary choice used)
  - random ($-calculus general choice used)
- Search can be
  - offline (n=0, the complete solution is computed first and executed after without perception)
  - online (n≠0, action execution and computation are interleaved.

On Problem Solving as an Instance of Multiobjective Minimization:

Given an objective/cost function $\$:A\times X \rightarrow R$, A is an algorithm operating on its input X and R set of real numbers, problem solving can be understood as a multiobjective (total) minimization problem to find $a^*\in A_F$ and $x^*\in X_F$, $A_F\subseteq A$ terminal states of algorithm, and $X_F\subseteq X$ terminal states of X such that $\$(a^*,x^*) = \min\{\$_1(\$_2(a),\$_3(x)),\ a\in A,\ x\in X\}$, where $\$_3$ is a problem-specific cost function, $\$_2$ is a search algorithm cost function, and $\$_1$ is an aggregating function combining $\$_2$ and $\$_3$.

- If $\$_1$ becomes an identity function we obtain *Pareto optimality* keeping objectives separate.
- For *optimization* (best quality solutions) - $\$_2$ is fixed, and $\$_3$ is used only.
- For *search optimization* (minimal search costs) - $\$_3$ is fixed, and $\$_2$ is used only.
- For *total optimization* (best quality solutions with min search costs) - both $\$_1$, $\$_2$ and $\$_3$ are used.
- kΩ-optimization (meta-search) - a very general search method that builds dynamically optimal or "satisficing" plans of actions from atomic and complex $-expressions.
- kΩ-meta-search is controlled by parameters:
  - k - depth of search
  - b - width of search
  - n - depth of execution
  - Ω - alphabet for optimization
- kΩ-optimization works iterating through phases: *select, examine, execute.*
- It is very flexible and powerful method: combines the best of both worlds: *deliberative* agents for flexibility, and *reactive* agents for robustness.

- The "best" programs are the programs with minimal cost - each statement in the language has its associated cost \$ (this leads to a new paradigm – *cost languages*).

Cost Performance Measures and Standard Cost Function:

- \$-calculus is built around the central notion of cost. The cost functions represent a uniform criterion of search and the quality of solutions in problem solving. Cost functions have their roots in von Neumann/Morgenstern utility theory and they satisfy axioms for utilities [6]. In decision theory they allow to choose states with optimal utilities on average (the maximum expected utility principle).
- In \$-calculus they allow to choose states with minimal costs subject to uncertainty (expressed by probabilities, fuzzy set or rough set membership functions).
- It is not clear whether it is possible to define a minimal and complete set of cost functions, i.e., possible to use "for anything". \$-calculus approximates this desire for universality by defining a standard cost function possible to use for many things (but not for everything, thus a user may define own cost functions)

3. Application of Information Theory Entropy as an Instance of \$-Calculus Cost Measure

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees, i.e., the ID3 algorithm by Quinlan and its successor C4.5 [6]. ID3 performs a simple hill-climbing search through the hypothesis space of possible decision trees using as an evaluation function the Shannon-based information gain measure. The information gain *Gain(S,A)* of an attribute *A* relative to a collection of examples *S* is defined as

$Gain(S,A) = Entropy(S) - \Sigma_{v \in Values(A)} Entropy(S_v)\ |S_v|/|S|$, where $Entropy(S) = \Sigma_i -p_i \log_2 p_i$.

Let's consider problem solving (learning + classification) for ID3 expressed as a special case of kΩ-search finding the shortest classification tree by minimizing the sum of negative gains, i.e., maximizing the sum of positive gains. The system consists of one agent , i.e., p=1, which is interested only in information gain for alphabet $A=\{a_i, a_{ij}\}, i,j$=1,2, i.e., Ω=A, i.e., costs of other actions are ignored (being neutral - in this case having 0 cost), and it uses a standard cost function $\$=\$_3$, where $\$_3$ represents the quality of solutions in the form of cumulative negative information gains - payoff in \$-calculus. In other words, total optimization is not performed - only regular optimization like in the original ID3. A weak congruence is used. In other words, empty actions have zero cost. The number of steps in the derivation tree selected for optimization in the examine phase k = 2, the branching factor $b = \infty$, and the number of steps selected for execution in the examine phase n = 0, i.e., execution is postponed until learning is over. Flags *gp*=*reinf*=*update*= *strongcong*= =0. The goal of learning/classification is to minimize the sum of negative information gains. The machine learning takes the form of the tree of \$-expressions that are built in the select phase, pruned in the examine phase and passed to execution phase for classification work. Data are split into training and test data as usual.

Let's assume for simplicity that we have only one decision attribute and two input attributes a_1 and a_2 with data taking two possible values on them denoted by $a_{11}, a_{12}, a_{21}, a_{22}$. Let's assume that cost of actions is equal to entropy of data associated with this action, i.e., $\$(a_i) = Entropy(a_i)$, $\$(a_{ij}) = Entropy(a_{ij})$, i, j= 1,2.

OPTIMIZATION: The goal will be to minimize the sum of costs (negative gains).

0. t = 0, initialization phase *init*: $S_0 = \varepsilon_0$

The initial tree consists of an empty action ε_0 representing a missing classification tree of which cost is ignored (a weak congruence). Because S_0 is not the goal state, the first loop iteration consisting of select, examine, and execute phase replaces an invisible ε_0 two steps deep (k=2) by all offsprings $b = \infty$.

1. t = 1, first loop iteration:

select phase *sel*:

$\varepsilon_0 = (? (. a_1 (+ (. a_{11} \varepsilon_{11}) (. a_{12} \varepsilon_{12}))) (. a_2 (+ (. a_{21} \varepsilon_{21}) (. a_{22} \varepsilon_{22}))))$

examine phase *exam*: $\$(S_0) = \$(\varepsilon_0) = \min (\$(a_1) + p_{12}\ \$(a_{12}), \$(a_2) + p_{22}\ \$(a_{22}))$

Let's assume that attribute a_1 was selected, i.e., \$-expression starting from a_1 is cheaper. Note that due to appropriate definition of the standard cost function [2,3] this is a negative gain from ID3.

Note that no estimates of future solutions are used (weak congruence - greedy hill climbing search). Execution is postponed (n = 0), and follow-up ε_{11} and ε_{12} will be selected for expansion in the next loop iteration.

2. t = 2, second loop iteration:

select phase *sel*: $\varepsilon_{11} = (. a_2 (+ (. a_{21}\ \varepsilon_{21}) (. a_{22}\ \varepsilon_{22})))$

Let's assume that ε_{22} has data from one class only, thus this is the leaf node - no further splitting of training data is required.

examine phase *exam*: nothing to optimize/prune - all attributes were used in the path or the leaf node contained sample data from one class of the decision attribute. Thus the end of the learning phase and the shortest decision tree is designated for the execution:

execute phase *exec*:

Test data are classified by the decision tree left from the select/examine phases. After that the kΩ-search re-initializes for the new problem to solve.

Note that we can change for example values of k (considering a few attributes in parallel), b, n and optimization to total optimization, then this will be related, but not ID3 algorithm any more. This is the biggest advantage and flexibility of \$-calculus automatic problem solving. It can modify "on fly" existing algorithms and design new algorithms, and not simulation of ID3 alone.

4. Conclusions and Future Work

Everything what we were able to demonstrate so far in this paper is that entropy can be used as a special case of \$-calculus cost function, however it cannot replace all instances of cost functions. One of the main unsolved problems of \$-calculus is axiomatization of cost functions in the style of Kolmogorov axiomatization of probability theory [6] (this might be an undecidable problem), or to estimate how good is approximation of all cost functions by \$-calculus standard cost function. We hypothesize that Kolmogorov complexity known also as algorithmic entropy [5] can be used to standardize \$-calculus search (algorithm) cost function $\$_2$, but this is left for future work.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Burgin, M. Super-recursive Algorithms, New York, Springer, 2005.
2. Eberbach E., Approximate Reasoning in the Algebra of Bounded Rational Agents, Intern. Journal of Approximate Reasoning, vol.49, issue 2, 2008, 316-330 (doi: dx.doi.org/10.1016/j.ijar.2006.09.014).
3. Eberbach E., The $-Calculus Process Algebra for Problem Solving: A Paradigmatic Shift in Handling Hard Computational Problems, Theoretical Computer Science, vol.383, no.2-3, 2007, 200-243 (doi: dx.doi.org/10.1016/j.tcs.2007.04.012).
4. Eberbach E., $-Calculus of Bounded Rational Agents: Flexible Optimization as Search under Bounded Resources in Interactive Systems, Fundamenta Informaticae, vol.68, no.1-2, 2005, 47-102.
5. Kolmogorov, A.N. (1965). On Tables of Random Numbers. *Sankhyā Ser. A.* **25**: 369–375.
6. Russell S., Norvig P., Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995 (3rd ed. 2010).
7. Shannon, C.E.; A mathematical theory of communication, *Bell Systems Technical Journal*, 1948, 27, 379-423, 623-656.