A Proposal of a Computational Device Based on Adaptive Automata

Ricardo Luis de Azevedo da ROCHA
Faculdade de informática (FCI/FEI)
Av. Humberto Castelo Branco 3972, Bairro Assunção
ZIP CODE 09850-901 – São Bernardo do Campo Brazil
e-mail: Ricardo-Rocha@acm.org

João JOSÉ NETO
Escola Politécnica da Universidade de São Paulo
Av. Luciano Gualberto, trav. 3, n.158 Cidade Universitária
ZIP CODE 05508 – São Paulo – Brazil
e-mail: joao.jose@pcs.usp.br

Abstract. We used an automated problem solution selection method, defined in [4], to build and study a computational device. The choice of solutions is carried out through a comparison among the adaptive automata models generated, based on the complexity of the models; the least complexity leads to a better model. We input a problem in grammar form. We also study the computational complexity of the device developed. The generated models are adaptive, which means that the formal substratum used in the method and the device is the adaptive automaton [1].

1. Introduction

In this section some definitions are put out, and their consequences are studied, preparing for the next sections, where we try to show that under some conditions the answer will be achieved in polynomial time [4], [5].

The results obtained here have some similarities with those results obtained by Valiant for PAC-learning [3, 12].

Definition 1: *A device BSMA (Adaptive Machine to Search for Solutions) is specified through the n-tuple* $M = (L_e, L_s, L_{mem}, E, \lambda, \kappa, \theta, \tau, \omega, \psi)$, where:

L_e: input language
L_s: output language
L_{mem}: work language, at the memory of the device
E: input function
λ : Combinations generation function
κ : Copy generation function
θ : Control function
τ : Measuring function
ω(n, m) : attachés' vector (constructions matrix), for each one of the n attachés, there is a maximum of m constructions
ψ : Output function.

In order to identify each construction, the ω(n, m) indicates for each attaché (1..n), m automata constructions. This way, each construction can be completely identified in the device, and used to generate new constructions.

Input Language: The input language of the BSMA device uses the situation logic [4], which allows receiving a specification that can be used to generate solutions. Each input should fulfill a situation, that is, indicating where, when and which are the actors of the situation. As established in [4], the input language contains the specification of the grammar for a problem, specified as a language, since we chose language ontology.

To characterize the device, we suppose that the input language allows an automaton specification, through the input function E. Therefore, starting from the input, the device deals only with automaton models.

Output Language: automata models generated by the device. Each model is composed by a list of transitions. The device answer is a list of the transitions of the automaton.

Memory Language: The device memory is composed by two infinite tapes: one unidirectional tape, completely filled with random positive integer values, and other, to work, which stores the computational steps, the computational cycles and the values caught by the measuring function. That tape is bi-directional, just to allow the access to the information regarding previous steps.

Definition 2: Control algorithm for the BSMA device. *This algorithm allows the coordination of the actions taken, activating constituent parts, and guaranteeing the search for solutions. If there is feasible solution, the algorithm 2 will drive to it.* The algorithm is sketched below.

```
BSMA-Control:
  Capture through the element Input the test data, the formulation of the initial model
  (hypothesis);
  Distribute the initial model by the constructions;
    For cycles = 1 until cycles-Limit do:
      Produce combinations among the constructions;
      Distribute to new constructions the new combined models;
      Induce random changes;
      For step = 1 until upper-Limit do:
        If step < upper-Limit then
          Perform a step in the constructions;
          Verify the state of the constructions;
          Update the attachés´ matrix;
      If there is a construction that didn't stop, mark it as inadequate;
      Exercise the other constructions;
      If there is a construction that doesn't drive to an answer, mark it as inadequate;
      Eliminate inadequate constructions;
      Store the appropriate constructions;
      Distribute the appropriate construction models to the constructions, until space-
      Limit beginning by the finalization order and by the smaller Kolmogorov
      complexity (using the measuring function $\tau$, definition 8);
    If there is no solution, change parameters of occupied space and maximum number of
    computation steps to reach “MaxLimit” and restart algorithm; (The value of MaxLimit
    is known by the algorithm, and it can vary in agreement with the implementation of the
    device)
  If there is no solution, indicate “no feasible solution”
    Else indicate “feasible solution” and the solution (chosen automaton model);
End BSMA-Control:
```

Definition 3: Internal state of the device configuration. Adapting the definition proposed by Shepherdson [7], *the internal state of the BSMA device represents a configuration of the device (in which each construction represents a Shepherdson processor), and it is represented by a quadruple* $\langle x, p_d, m, p_r \rangle$, in which we have:

- *x*: control state (current state), represented by the state of each construction j, inside of the attaché i. Then, x is an element of ω(i, j).
- p_d: processor dispatcher, finite group of tuples ⟨y, *f*, x_1, ..., x_n⟩, that represents, each one, the instruction to compute $f(\lambda_1, ..., \lambda_n)$, where λ_i are data elements located in x_1, ..., x_n, and y is the label of the processor, uniquely identified. For the BSMA device, tuples are composed by the transitions of the adaptive automaton model, which are particular to each construction. Thus, it is always possible to the controller element to determine what was the consequence of a construction computational step, and, to indicate which construction should accomplish the computation of a function.
- *m*: memory map, a finite group of orderly pairs ⟨ℓ, d⟩ indicating that the element described by d is located in the position ℓ. The memory of the BSMA device is composed by the states and transitions of the constructions, as well as of all the attachés, along the time. This means that, to each problem, it is possible to know if a certain construction inside of an attaché was previously used.
- p_r: processor record. The matrix ω(n, m) indicates the current state of each processing unit (construction), describing each one of the n automaton attachés and their internal constructions (M_1 .. M_m).

Definition 4: Computational step. *In the BSMA device, a computational step can take one of two ways: a complete transition that occurs in the constructions, even if it is needed to change the topology of an automaton model in a construction, or, a step in the controller element machine.* When the constructions (solution models) are being exercised inside of the device, a step is interpreted as a transition. Therefore, let α_k be a configuration of an automaton model of construction j of attaché i. After a computational step k in this construction, a transition occurs and changes internal configuration to α_{k+1}.
Thus: $\alpha_k \underset{BSMA_{\omega(i,j)}}{\vdash} \alpha_{k+1}$. The symbol $\underset{BSMA_{\omega(i,j)}}{\vdash}$ indicates a computational step performed by the BSMA device, through the construction ω(i, j).

Definition 5: Computational cycle. *In the BSMA device, a computational cycle is an amount of computation steps previously defined that configures a limit.* When some construction finishes it's processing in a smaller or equal number of steps than the limit of the cycle, we have a closed cycle of computation. Otherwise, the cycle is considered unfinished, and its model should be discarded.

Therefore, a computational cycle reaches a final configuration α_f from an initial configuration α_i. If α_f is the configuration that finishes the processing of the construction, then the cycle is said closed. So: $\alpha_i \overset{*}{\underset{BSMA_{\omega(i,j)}}{\vdash}} \alpha_f$. The symbol * represents the transitive and reflexive closure over the step operation, and is valid only if the cycle is closed.

Lemma 6: *On a computation cycle in a BSMA device with limit of steps $N_l \geq 0$, all the exercised automaton constructions conclude their computational process in a number of steps $n \leq N_l$.*

Proof: Immediate from the definitions of computation step and cycle.

Theorem 7: *A BSMA device, that has a limit of steps $N_l \geq 0$, avoids the "Turing machine halting problem".*

Proof: It follows directly from previous Lemma 6.

Definition 8: Measuring function to the BSMA device. τ - *(Function aim maintenance). This function allows task cost measuring of the relative distance of each construction to a possible solution.* However, it doesn't involve direct storage of time, it does an indirect measure, through the computational steps.

The relative distance is measured through the complexity of each construction. All these information are stored in the memory tape, in a way to modify the relative weight of the measures taken.

The measuring function is used only on the constructions that reached a final configuration. Its expression is similar to the induction principle MDL, carried out over the automaton model.

The invariance theorem [3] shows that different codes of the same algorithm lead to similar values of complexity, and that the difference obtained is only due to the code used for the conversion of the algorithm to a particular machine. Through that theorem, and the fact that an adaptive automaton model is equivalent to a Turing machine model [6], we can conclude that the complexity measure of the adaptive automaton model also represents the algorithm complexity of the program of the universal Turing machine that simulates it.

For the MDL principle [3], the derivation of an approximation for the Kolmogorov complexity measure results in: $L(x) = K(x) \approx \log(x) + 2\log(\log(x))$, for binary strings x. The complexity measure of an adaptive automaton model cannot be performed only in terms of the amount of states and transitions, but we must consider the transitions that contain activations of adaptive functions, which can increase or decrease the amount of states and transitions of the automaton at each computation step. We compute n in the following way: n = total number of states + sum of all the adaptive actions of increase + sum of all the adaptive actions of deletion + number of transitions. In the present case, L(n) represents the measure of the code length based on the number n calculated as described.

Definition 9: Actions control on the BSMA device (*support to the aim maintenance* function). *This function allows the controller to mediate the actions taken inside of the attachés. This function guarantees that the computational process is feasible.*

A computational process is considered feasible if it offers an answer within certain limits, in time and space. This characterization of feasible creates a dependency between outside parameters and the computational process, because waiting for an answer of a computational system is a function determined by the user of that system [3].

Thereby, we can define feasible computation as a task whose processing is limited by a control function. This function has as parameters a couple of arguments, the user's acceptable space and time limits.

During the computational process, the control function indicates if this process is within the imposed limits. If it is not, it will be stopped, because it escaped to the desired pattern. So, the foreseen computational limits are incorporated to the device.

Let θ be the control function. $\theta : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$, θ Maps pairs of natural numbers in pairs of natural numbers, with the following properties:

a) $\theta(1, 1) = (T, p + d)$, where "T" represents the time of the largest computational step performed by the device, "p" represents the space occupied by an adaptive transition in number of bits, on the device, and "d" represents the space occupied by the representation of the device in bits.

b) $\theta(m, 1) = (m \times T, p + d)$, where "m" represents the number of computational steps carried out by the device.

c) $\theta(m, z) = (m \times T, z \times p + d)$, where "z" represents the number of constructions used by the device.

Therefore, the feasible computation limits, in terms of space, can be determined through the knowledge of the maximum number of attachés. Assume such value as "a". Thus, supposing that the external values supplied for a feasible computation are time "t" and space "e", we have:

The maximum number of steps, n_{max}, is $n_{\max} = \left\lfloor \frac{t}{T} \right\rfloor$. Maximum space occupied by attaché A_{max}, maximum space occupied by construction C_{max}, $A_{\max} = \left\lfloor \frac{e-d}{a} \right\rfloor$, $C_{\max} = \left\lfloor \frac{A_{\max}}{p} \right\rfloor$.

As a consequence of these definitions, we can deal with feasible computation inside a computational model through the control function θ, which maps the external information inside the process.

Lemma 10: *A BSMA device satisfies to the four Gandy-Shepherdson principles for mechanisms computing over arbitrary structures* [7].

Proof: Based directly on the Gandy-Shepherdson principles and definition 3.

Theorem 11: *A BSMA device has a parallel Turing machine computational power* [7].

Proof: The computation over partial structures Theorem (see [7]), demonstrates that any mechanism that obeys the Gandy-Shepherdson principles is equivalent to a parallel Turing machine. Lemma 10 shows that the device obeys the principles.

Lemma 12: Prefix code Computation. *In the BSMA device, the computational process accomplished by the automaton constructions is based on a prefix code, which means that an automaton model represents a self-delimiting prefix code.*

Proof: By the definition of partially recursive prefix function. A self-delimiting prefix code is defined as a computational process that accomplishes a partially recursive prefix function.

According to the definition, found in [4] and in [1], an automaton model can finish its computation if, and only if, it reaches a final state. In any case, the model doesn't need any information about the next states or transitions. Therefore, by the definition, it constitutes a self-delimiting code. Each model represents a prefix, and this prefix is proper.

Let $\phi_i(x)$ be a partially recursive prefix function, that can be carried out by an automaton model $T_j(i, x)$. By Theorem 11, when granting to each parallel model the possibility of execution within limits of time and space, the BSMA device is, in fact, creating an enumeration of models $T_j(i, x)$, that are capable to complete their computation task in the smallest number of steps. Thus, the device defines an enumeration in the space of configurations, containing only self-delimiting prefix codes.

■

Lemma 13: Kolmogorov complexity. *The BSMA device implements an algorithm that allows the determination of a complexity measure distribution and of the smallest Kolmogorov complexity measure for the exercised models (established in the definition 8).*

Theorem 14: Search for solutions algorithm. *In the BSMA device, the implemented search for solutions algorithm follows the LSEARCH algorithm [3].*

Proof: From the Lemmas 12, 13, and definition 2, the device implements an enumeration of the automaton models that concluded their computation, and traverses the automaton models space through the insertion of probabilistic changes and combinations. The LSEARCH algorithm does exactly this task, but in lexicographical order. This way, the device implements the LSEARCH algorithm; even so it uses a random order distribution for the models, it searches the same space of models.

■

Definition 15: Function for generation of random sequences in the BSMA device. *This function allows the generation of a sequence of pseudo-random values. These values are numeric and they are stored in the memory tape of the device.*

Definition 16: Function for verification of random sequences in the BSMA device. *This function allows verifying if a sequence of values in the memory tape is random, through a Martin-Löf test of on the sequence.* The test consists of the verification of the value found for the frequency of occurrence of the symbols in the sequence, which should have equivalent value for each symbol [3] [9].

Lemma 17: Random sequences in the Device. *In the BSMA device, a random sequence can be used in the case of random induction, or combination.*
Proof: Immediate of the definitions 15, 16.

■

Lemma 18: Convergence of the expected value. *Let S_n be the expected value of the square of the difference between the value of an universal measure for the a priori probability and the probability value attributed through the distribution m(x) established in the Lemma 13, then* $\sum_n S_n \leq \frac{K(L)\ln 2}{2}$.
Proof: Based on the Lemma 13 and in the Expected Value Convergence Theorem for Turing machines, proved in [8] and [3].

■

Lemma 19: Space complexity in the BSMA device. *The occupied space by the BSMA device, in a feasible computation, is given by a function of order $O(n^p)$, in which "n" is the maximum number of states found in the generated models.*
Proof: Let C_{max} be the obtained value of the definition 9, the maximum space occupied by a construction, and make $C = a.C_{max}$, where a represents the number of attachés, according to the definition 9. Thus, supposing the existence of a positive integer value r, whose value is: $r = \left\lceil \frac{\log C}{\log n} \right\rceil$, where C and n were previously defined, we have: $n^r \leq C$.
We can suppose that we are allowed to have a new construction whenever necessary, until reaching a maximum number of C constructions. In those conditions, the space limit of each automaton model is of the order $O(n^2)$ [6], where n represents the number of states of the model. Admitting M as a total of models and letting E be the occupied space limit of an automaton model, we have that $M = {}^{C}/_{E}$. Therefore, $O(M) = \frac{O(n^r)}{O(n^2)} = O(n^{r-2})$. Making then $p = r–2$, we have thereby that the occupied space in a feasible computation is limited by $O(M) = O(n^p)$, that is a function limited by the order $O(n^p)$.

■

Theorem 20: Time complexity, for the BSMA device. *The time spent by the BSMA device to find a feasible solution, in a single group of computational cycles (using the original parameters of the device), is given by a function of the order O[n.t(n)] where "n" is the size of the largest string of a group of strings to be analyzed, and t(n) is time that another algorithm takes to invert the problem.*
Proof: Based on Theorem A.8 [6], since the adaptive automaton spends time proportional to a Turing machine and, in this case, it is proportional to the size of the string "*n*" for each construction. Let "m" be the number of strings of the set to be analyzed. Despite of having to test "m" strings, the device can make it in a concurrent way (supposing that doesn't overcome the limits of the previous definition 9). By the LSEARCH Theorem of [3], we have that, if a function inverts the problem in *t(n)*, then the algorithm inverts it in C.t(n*)* and, as the device implements the LSEARCH algorithm, it also inverts the problem in C.t(n*)*, that is, of the order *O[t(n)]*. Even so, by the hypothesis, the number of computation cycles c was not exceeded. Thus, if $k \geq {}^{m}/_{n}$ is a constant integer value, representing the relationship between the number of strings and the largest string of the group, we have approximately *t(n)* × m steps, or *t(n)* × (k.*n*) steps. This result is only valid if the solution was found with the original parameters. Therefore, the time spent by the device searching for a feasible solution is of the order *O[n.t(n)]*.

■

Definition 21: Function to generate copies in the BSMA device. *This function allows generating copies of the constructions present in the attachés, in a way to allow that each*

construction can be used again, in its original formulation before the computation cycle have been initiated.
<u>Definition 22:</u> Function to control the changes in the BSMA device (combination function). *This function allows the controller to ask for changes by combination, on the constructions inside the attachés.* The combination is always accomplished among neighboring constructions, through the exchange of a transition. The choice of the transitions to be exchanged is random, and is indicated by the controller that supplies the number of the transition, and indicates which are the constructions that must exchange.
<u>Definition 23:</u> Output function supplied by the BSMA device. *This function allows the controller of the device to output the current constructions in the attachés.* In the control algorithm, the appropriate outputs are candidates to the solution. However, the chosen solution is the one that exhibits the smallest complexity, and their transitions are sent, exactly as stored internally.
<u>Definition 24:</u> Evident solution in the BSMA device. *A solution found by the BSMA device is evident if the amount "Tm" of models internally generated until finding it is: $Tm \leq n^k$, where n represents the maximum number of states of the generated models and $k \geq 1$.* This means that a solution is considered evident, if the number of necessary models to find it is limited by a polynomial order, in relation to the maximum number of states of the models.
<u>Theorem 25:</u> *To a solution of a problem that can be found in the BSMA device, we admit that each generated model in the device can answer in polynomial time and have a maximum number of states n, so we have that the time complexity to find the solution is of the order $O(|\Sigma|^{n^2})$, where $|\Sigma|$ represents the amount of symbols of the alphabet used by the device. In the cases where the solution model is evident, this value can be reduced in terms of time to O[p(n)], that is, polynomial time.*
<u>Proof:</u> Using the adaptive automaton definition, we observe that there are a number of symbols $|\Sigma|$. Therefore, each possible state can consume different $|\Sigma|$ symbols. An automaton model, with *n* maximum number of states, occupies space corresponding to the order $O(n^2)$, that is to say, the number of possible transitions is of the order $O(n^2)$. Therefore, there are $O(|\Sigma|^{n^2})$ automaton models to be generated and tested. By the hypothesis, each model answers in polynomial time. We have then that the time complexity is of the order $O(|\Sigma|^{n^2})$.

For the cases in that is evident solution, the definition 24 establishes that we need to generate and test only an amount of models of the order $O(n^k)$, where $k \geq 1$, to find a solution. Let *q(n)* be a polynomial that represents the necessary order of time to evaluate each model – hypothesis of answer in polynomial time –, then the time *t(n)* necessary to evaluate each one of the models is equivalent to the order $O[q(n).n^k]$, because we need to evaluate the whole search space. To the generation of the models, the time complexity in terms of time spent by the device, is at most of the order *O[t(n)]*. Therefore, as the generation and the evaluation are accomplished in sequence, we have that the complexity order for the task of finding a solution is at most of $O[q(n).n^k]$. So, if the solution is evident, the complexity reduction is from the order $O(|\Sigma|^{n^2})$ to the order *O[p(n)]*, where $p(n) = q(n).n^k$.

■

<u>Theorem 26:</u> *To a problem that can be represented in the BSMA device, with evident solution, which feasible computation space-complexity is of the order $O(n^p)$ (by Lemma 19) inside the device, we have: if $k \leq p$ the solution in a process of feasible computation will be optimum, and, if $k > p$, we cannot guarantee that if we find a solution it's an optimum one.*
<u>Proof:</u> Using the definition 24 and the theorem 25, the amount of necessary generated models to find a solution is of the order $O(n^k)$.

Lemma 19 establishes that a feasible computation occupies space of the order $O(n^p)$, that is to say, it generates an amount of constructions of the order $O(n^p)$.

By hypothesis we have a feasible computation for the case of $k \leq$ p. Therefore, the optimum solution is inside the space to be searched, thus, the device finds an optimum solution in polynomial time.

In the case where $k >$ p, and, by hypothesis, there is feasible computation, then we can find a solution. However, as the search space $O(n^p)$ can be lower than the necessary space to find the optimum solution, there is no warranty that the optimum solution can be found in a feasible computation.

■

5. Final Remarks

For problems that involve optimization, or search for the best alternative, the BSMA device proposed is propitious, since it always accomplishes the search for the best solution model, based on complexity reduction. We need to formulate the problem as a complexity minimization problem expressed in language form, and then use the device.

Finally, in the BSMA device we can have computational solution in polynomial time, even when k > p (Theorem 26), or when there is no evident solution, although there is no warranty that it can be found. Finding such solutions (non evident solution) in polynomial time depends on the organization of the search space, because if there is a solution that can be found in the n^p limit, then it will be found, although it is not evident. The solutions demonstrated as being evident, with k ≤ p, are found in polynomial time.

These results are consistent and quite similar to the results obtained by Valiant for PAC-learning [3, 12], but less restrictive in the sense that even though we cannot guarantee that a solution would be found, it can be found. Which indicates that these results can be generalized.

References

[1] J. José Neto, Adaptive automata for context-dependent languages. *ACM SIGPLAN Notices*, v. 29, 9, 1994, pp. 115-124.

[2] H. R. Lewis; C. H. Papadimitriou, *Elements of the theory of computation*. New Jersey, Prentice-Hall, Inc, 1998.

[3] M. Li; P. Vitányi, *An introduction to Kolmogorov complexity and its applications*. 2nd. ed., New York, Springer-Verlag, 1997.

[4] R. L. A. Rocha, *Um método de escolha automática de soluções usando tecnologia adaptativa*, Doctoral Dissertation – Polytechnic School, Universidade de São Paulo, São Paulo, 2000. 211p.

[5] R. L. A. Rocha; J. J. Neto, Uma proposta de método adaptativo para a seleção automática de soluções. VI Congreso Internacional de Ingeniería Informática – ICIE, *Proceedings*, ISBN: 98 7461 764 7, Buenos Aires, Argentina, April, 2000.

[6] R. L. A. Rocha; J. J. Neto, *Autômato adaptativo, limites e complexidade em comparação com a máquina de Turing*. LAPTEC 2000, ISBN: 85 8579 529 8,São Paulo, Brazil, 2000, pp. 33-48.

[7] J. C. Shepherdson, Mechanisms for computing over abstract structures. In: HERKEN, R. ed. *The universal Turing machine*. New York, Springer-Verlag, 1995, pp. 537-555.

[8] R. J. Solomonoff, Complexity based induction systems: comparisons and convergence theorems. *IEEE Transactions on Information Theory*, v. IT-24, n.4, pp.442-432, 1978.

[9] Y. Wang, *Randomness and complexity*. Heidelberg, Doctoral Dissertation - Naturwissenschaftlich-Mathematischen Gesamtfakultät, Ruprecht-Karls-Universität, 1996. 104 p.

[10] S. J. Russell; Norvig, P. *Artificial intelligence a modern approach*. New Jersey, Prentice-Hall, Inc., 1995.

[11] R. J. Solomonoff, A formal theory of inductive inference - Part I and Part II. *Information Control*, v.7, 1964, pp.1-22; pp.224-254.

[12] L. G. Valiant, A theory of learnable. *Communications of the ACM*, v.27, n.11, 1984, pp.1134-1142.