Optimization of state assignment in a finite state machine: evaluation of a simulated annealing approach

Reinaldo da Silva Ribeiro[1], Rafael Lima de Carvalho[1] and Tiago da Silva Almeida[1,2]

[1] *Universidade Federal do Tocantins, Department of Computer Science, Palmas / TO, Brazil*
[2] *State University of Campinas, Institute of Computing, Campinas / SP, Brazil*

Reception date of the manuscript: 18/08/2021
Acceptance date of the manuscript: 16/11/2021
Publication date: 17/11/2021

Abstract— In this research, the application of the Simulated Annealing algorithm to solve the state assignment problem in finite state machines is investigated. The state assignment is a classic NP-Complete problem in digital systems design and impacts directly on both area and power costs as well as on the design time. The solutions found in the literature uses population-based methods that consume additional computer resources. The Simulated Annealing algorithm has been chosen because it does not use populations while seeking a solution. Therefore, the objective of this research is to evaluate the impact on the quality of the solution when using the Simulated Annealing approach. The proposed solution is evaluated using the LGSynth89 benchmark and compared with other approaches in the state-of-the-art. The experimental simulations point out an average loss in solution quality of 11%, while an average processing performance of 86%. The results indicate that it is possible to have few quality losses with a significant increase in processing performance.

Keywords— Finite State Machine. Simulated Annealing. Digital Systems. Metaheuristic.

I. Introduction

Hardware optimization demands a lot of research, not only from the professionals responsible for developing the hardware but also from physicists and chemists who can find ways or elements that improve the functioning of the hardware.

Regardless of the application, every hardware project is complex, demanding a great intellectual effort and, consequently, monetary. The process of design and development of a hardware component has several steps. In this work, the focus is on the optimization of Finite State Machines (FSM).

FSMs are abstractions of the behavior of a given circuit, whether it is a part of the whole of an Application-Specific Integrated Circuit (ASIC) or a conventional processor. When thinking in terms of algorithm, it is referred to the sequence of commands, or steps, that in a certain order performs a task.

This algorithm can be abstracted in the form of a machine where each step is represented by a state. Conventional computers allow us to perform only one step at a time and the transition between states is made through external or internal stimulus (inputs).

From this representation, it is possible to provide a physical model, where this FSM model can be synchronous about to with concerning the internal or external behavior, as well as it can be asynchronous, varying according to the application.

The optimization of an FSM can lead to a reduction in the physical size of the final circuit, resulting in savings in the critical path, area, and power. For the optimization of the FSM, the goal is composed of finding the best allocation of states and minimizing the size of the Boolean expressions that represent the machine behavior.

This is not a recent research topic [1, 2], however, due to its importance and being an NP-Complete[3] problem, is still an open topic because of breaking down of Dennard's law [4], which states that as the dimensions of a device go down, so does the power consumption. And many complex metaheuristic algorithms have been tested for this problem, such as Evolutionary Algorithms [5, 6], Tabu Seach [7], and Simulated Annealing [8, 9, 10].

As far as it is known, Ahmad et al [10] have proposed a complex hybrid method combining Genetic Algorithms with Simulated Annealing, to find optimal state-machine allocations. Thus, arises the question **how distant is the result with a much simpler and faster metaheuristic, which uses less computational resources (without a population of solutions)?** Therefore, the objective of this investigation is to provide an answer to this question. Furthermore, the main contribution of this paper is **the evaluation of the state assignment in a finite state machine solution produced by a simulated annealing algorithm**.

The remaining text is organized as the basic definitions about the considered problem are presented in Section IIa.

Contact data: Reinaldo da Silva Ribeiro, reinaldorib28@gmail.com

Furthermore, the Simulated Annealing is reviewed in Section IIb. The experimental setup, as well as the SA algorithm, is presented in Section III. In addition, the results are discussed in Section IV. Finally, the conclusions and future research directions are shown in Section V.

II. DEFINITIONS

a. Finite State Machine

Sequential circuits can be defined as circuits with a section made of combinational logic and another section of memory which are normally flip-flops. Where each stage that the sequential circuit advances are called a state. In each state, the circuit stores the inputs passed to define its output, and the state transition only occurs with the clock variation [11, 12].

An FSM has a finite number of inputs, constituting the set of $N = \{N_1, N_2, ..., N_n\}$. Thus, the circuit has a finite number of outputs, determined by the set of $M = \{M_1, M_2, ..., M_m\}$. The value contained in each memory element is called state variables, forming the set of $K = \{K_1, K_2, ..., K_k\}$. The values contained in the K memory elements define the current state of the machine. The internal transition functions generate the next state set $S = \{S_1, S_2, ..., S_s\}$, which depend on the inputs N and the current states K of the machine and are defined through combinational circuits . The values of S, which appear in the state machine transition function at time t, determine the values of the state variables at time $t+1$, and therefore define the next state of the machine.

The behavior of an FSM can be described through a state transition diagram or a state transition table. A state transition diagram or state transition table lists the current state, next state, input, and output. A state transition table has 2^N columns, one for each occurrence of the input set and 2^K rows, one for each occurrence of the state set.

The transition diagram is an oriented graph, where each node represents a state, and from each node emanate p oriented edges corresponding to the state transitions. Each oriented edge is labeled with the input that determines the transition and the output generated. FSM determine the next state $K(t+1)$, based only on the current state $K(t)$ and the current input $N(t)$. FSM can be represented by,

$$K(t+1) = f[K(t), N(t)] \tag{1}$$

where f is a state transition function. The output value $M(t)$ is obtained by,

$$M(t+1) = g[K(t)] \tag{2}$$

$$M(t+1) = g[K(t), N(t)] \tag{3}$$

where g is an output function.

An FSM with properties described in the Eqs. (1) and (2) is called a *Moore* Machine and a machine described through the Eqs. (1) and (3) is called the *Mealy* Machine.

The operation of computers is based on the operation of transistors, which depending on the amount of stored charge, the signal can be interpreted as high (1) or low (0), and off (no stored energy).

As the computer works on the interpretation of two electrical impulses can be observed that it is a binary system, therefore, being governed by Boolean algebra.

Boolean algebra is an algebraic structure that defines the arithmetic of logical operators that, being composed of the symbols $S = \{0, 1\}$, constitute a binary system. The concepts of Boolean algebra are also used in electronics since physical circuits are rather designed in abstractions, called logic circuits.

Given a Boolean space, a *variable* is a symbol representing a coordinate in that space. A variable or its negation is called *literal*. The term product is defined as the Boolean product of one or more literals. A minimal term, or *minterm*, is a term product that outputs a value '1'. A circuit with all variables in certain cases can be simplified, eliminating redundancies and having its size reduced. A Boolean function that implies a combination of minterms is called the *implicant* of a function, and an implicant that cannot be reduced, that is, does not imply another function, is called a *prime implicant*. The sum of all implicants and prime implicants of a function is the set of minterms for which the function's result is '1'.

When representing an FSM, usually are used words or letters to refer to states, since the number of flip-flops needed to represent an FSM is calculated similarly to the number of rows in the truth table. When assigning a value to a state, each literal symbolizes the value that will be delivered to a respective flip-flop at a given time. *The joining of the values of each flip-flop is equivalent to the value assigned to a given state of the FSM*.

The values present in the memory element, when combined, represent the current state. The flip-flops are then connected in combinational circuits that change the value contained in the flip-flop at each clock pulse, making the flip-flops start to represent the value assigned to the next state of the machine, going from the current state to the next state.

The combinational circuit responsible for this change of states is the result of simplifying the expressions obtained from the inputs of a given flip-flop and the stimulus that will be given. The circuit receives the flip-flop output value and the machine state stimulus value. The set of output values represent the next state that the state machine will assume.

The state assignment is fundamental when there is the intention to optimize, as it is directly linked to the size of the expression that will make the change between the current state and the next state. Changing the distribution of values drastically affects the size of the expression, which consequently increases the size of the circuit.

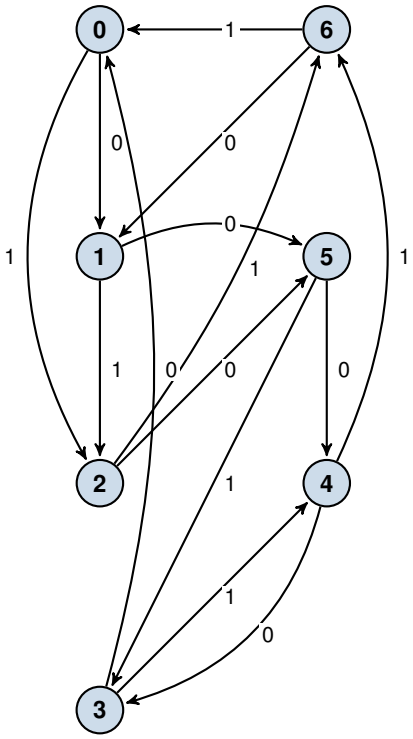
For instance, a 7-state FSM, where the assignment of values to the states is done sequentially, from 0 to 6 with numbers on a binary basis. Table 1 shows the arrangement of assigning values to states. The state transition diagram is depicted in Fig. 1.

The graphical representation of the state transition diagram can also be expressed by the *state transition table* with the transitions as a function of the inputs, as shown in Fig. 1, with the state assignment of the Table 1. In Table 2, where the flip-flops are represented by the variables Q_2 Q_1 Q_0, and d represents the input value that the state will receive. The expressions that generate the value of the next state are given

TABLE 1: FIRST ASSIGNMENT.

State	Assignment
0	000
1	001
2	010
3	011
4	100
5	101
6	110

Fig. 1: Example of a state transition diagram for an FSM.



by Y_2, Y_1, and Y_0.

As the FSM has seven states, it can be represented by three flip-flops, and with the addition of the input, therefore four is the minimal number of values needed to represent the combination of inputs necessary to define the transitions. As a result, the truth table has sixteen rows.

To obtain the expressions, the Karnaugh map simplification method was used, which facilitates the grouping of terms to perform the operations that allow reducing the expression, as illustrated in Fig. 2.

The Karnaugh map is used to simplify and find the respective logical expression for each Y_i. The simplified expression serves as the basis for the construction of the corresponding logical circuit. With this simplification, its obtained:

$$Y_2 = \bar{Q}_2 Q_1 \bar{Q}_0 + Q_1 Q_0 d + \bar{Q}_1 Q_0 \bar{d} + Q_2 \bar{Q}_1 \bar{Q}_0 d$$
$$Y_1 = \bar{Q}_2 \bar{Q}_0 d + Q_2 \bar{Q}_1 \bar{Q}_0 + \bar{Q}_1 d$$
$$Y_0 = \bar{Q}_0 \bar{d} + \bar{Q}_2 \bar{Q}_1 d + Q_2 Q_0 d$$

Performing a new assignment of values to states randomly, instead of doing it sequentially or ordered, can result in several possible combinations, one of which was chosen and represented in Table 3.

TABLE 2: TABLE REFERRING TO THE FIRST ASSIGNMENT OF STATES.

Q_2	Q_1	Q_0	d	Y_2	Y_1	Y_0
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	1	0	1
0	0	1	1	0	1	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	0	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	0
1	0	1	1	0	1	1
1	1	0	0	0	0	1
1	1	0	1	0	0	0
1	1	1	0	X	X	X
1	1	1	1	X	X	X

TABLE 3: SECOND ASSIGNMENT OF STATES FOR THE EXAMPLE.

State	Assignment
0	010
1	101
2	000
3	110
4	001
5	011
6	100

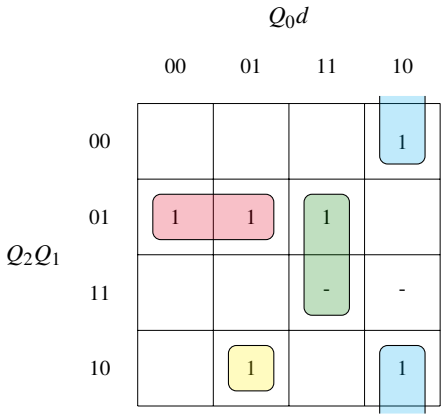
The resulting FSM in the new assignment has the same graph structure and transitions, but with different values assigned to each state. The expression that will be obtained with the simplification makes the resulting circuit different from the previous state assignment. The state transition table referring to the FSM after the new assignments is shown in Table 4.

TABLE 4: TABLE REFERRING TO THE SECOND ASSIGNMENT OF STATES.

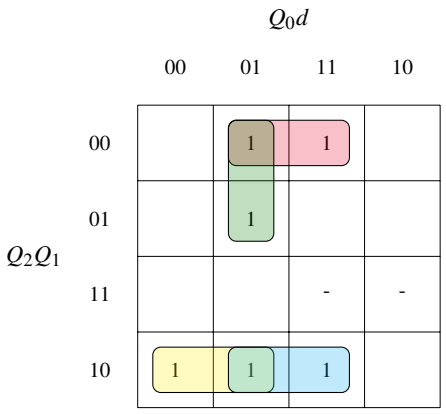
Q_2	Q_1	Q_0	d	Y_2	Y_1	Y_0
0	1	0	0	1	0	1
0	1	0	1	0	0	0
1	0	1	0	0	1	1
1	0	1	1	0	0	0
0	0	0	0	0	1	1
0	0	0	1	1	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
0	0	1	0	1	1	0
0	0	1	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	1	1	0
1	0	0	0	1	0	1
1	0	0	1	0	1	0
1	1	1	0	X	X	X
1	1	1	1	X	X	X

With the new assignment, it is possible to see from the

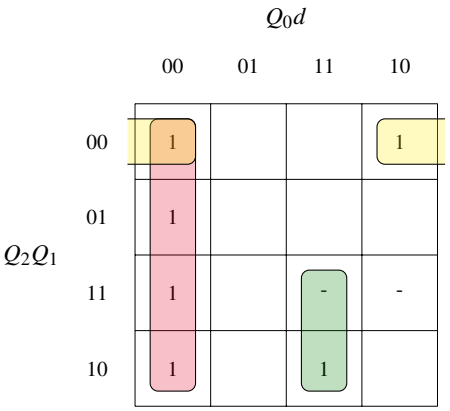
Fig. 2: Karnaugh map to obtain boolean expressions for Y_2, Y_1 e Y_0.



(a) Map of Y_2



(b) Map of Y_1



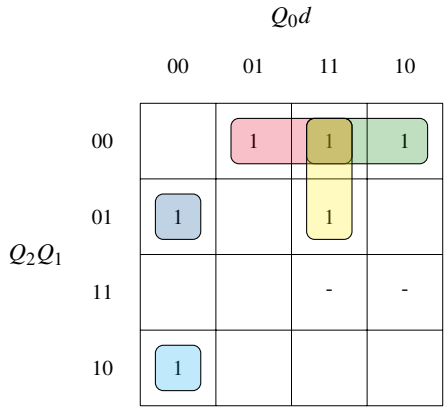
(c) Map of Y_0

resulting Karnaugh maps, shown in Fig. 3, that there will be no minterms of two variables. With the simplification we obtain:

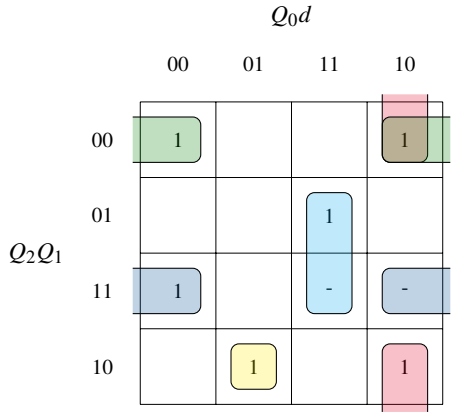
$$Y_2 = \bar{Q}_2Q_1\bar{Q}_0\bar{d} + Q_2\bar{Q}_1\bar{Q}_0\bar{d} + Q_1Q_0d + \bar{Q}_2\bar{Q}_1d + \bar{Q}_2\bar{Q}_1Q_0$$
$$Y_1 = \bar{Q}_2\bar{Q}_1\bar{Q}_0\bar{d} + Q_2Q_1\bar{Q}_0\bar{d} + Q_2\bar{Q}_1\bar{Q}_0d + \bar{Q}_1Q_0\bar{d} + Q_1Q_0d$$
$$Y_0 = \bar{Q}_2\bar{Q}_0\bar{d} + \bar{Q}_1\bar{Q}_0\bar{d} + Q_2Q_1d + Q_1Q_0\bar{d} + Q_2Q_0\bar{d}$$

The result of the simplifications shows that there was a change in the size of expressions. This size difference in an FSM with many states can be drastic, causing a considerable increase in power consumption, physical circuit size, and execution time.

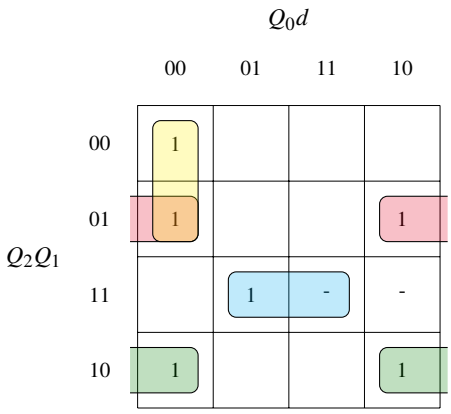
Fig. 3: Karnaugh map to obtain Boolean expressions for Y_2, Y_1 e Y_0 with the second assignment.



(a) Map of Y_2



(b) Map of Y_1



(c) Map of Y_0

b. Simulated Annealing

SA is a technique that simulates the heating and cooling process of materials, allowing the escape of optimal locations, and a better exploration of the search space [13, 14]. When a certain material is heated, there is the excitation of the molecules, and how it cools down to the stability point of the molecules will determine characteristics such as hardness, strength, flexibility, etc.

The way a metal reacts to stress is directly related to how its micro-structure is organized. The capacity of a material deforms under stress is called **Plastic Deformation**[15]. The deformation of a material occurs through rearrangement of the molecules that constitute the crystal grain, a large crystal grain has more molecules to shift during the applying of forces. Since the limit of deformation is related to the size of the grain, a material composed of small crystal grains, when subjected to stress, won't deform as much as a material with large crystal grains.

The annealing process through slow cooling makes the micro-structure of the finished product composed mostly of large crystals, making the material softer, therefore more susceptible to plastic deformation.

In annealing, the metal is heated to a uniform temperature throughout its length, and allowed to cool slowly, gradually, and uniformly. Thus, giving the material a better structuring and organization of the material's molecules, resulting in a more flexible material. The annealing allows the material to be soft, which is better for molding and electrical conductivity.

In the tempering of metal, the material is heated to a temperature close to melting point and cooled abruptly, causing the micro-structure to stabilize with small crystals, with little leeway for deformations, making the material hard.

The computation uses the idea of temperature, which slowly drops to the point of stability. SA as the search and improvement algorithm of Hill Climbing uses this temperature parameter to control when jumps out of the optimal locations occur [16, 17, 13].

The algorithm works as follows: an initial solution is defined, from which the algorithm will start, and a value that will behave like the temperature in thermodynamics, and will decrease in small steps, as in slow cooling. Within each iteration of temperature parameter reduction, another routine of defined size occurs, where another random solution is generated within the search space that will be compared with the initial solution.

This internal routine occurs n times within a temperature adjustment step. In this way, seeking stability of the solution within the temperature range in which the algorithm execution is located. As the temperature parameter decreases, this routine starts to be executed with fewer chances of a new solution being chosen, due to the acceptance criteria [16, 17, 13].

The acceptance of a new solution is based on a thermodynamic model in which starting from a system state i of energy E_i, a new state j of energy E_j is generated based on a permutation.

If the energy difference between current state (i) and new state (j) is greater than zero, the new state is chosen. If the difference is less than zero, the probability that the state j replaces i and becomes the current state is given by[14]:

$$P(\Delta E, T) = e^{\left(-\frac{-(E_j - E_i)}{T}\right)}, \tag{4}$$

where T represents the current temperature.

At the beginning of the algorithm execution, where the temperature is high, there is a greater probability that fewer local solutions will be accepted, promoting the leap to other parts of the search space. But as the temperature drops, this possibility is also present, suitable as chances of jumping to distant points in the search space [16, 17, 13].

III. Methodology for the Experiments

The classic model of the SA algorithm was explained in the previous section, but how certain parts of the work were organized required that the structure of the algorithm be changed to better suit the problem.

How the search space is created for the state assignments for the FSM is random and it is not controlled by any parameter, so there is no way to control whether a solution is in the local neighborhood or a distant point of the space search, or even control the distance of the jump depending on the temperature.

For this reason, the SA applied to the problem only accepts new solutions if the quality is better than the current one, and the probability is favorable. The algorithm works like a Hill Climb where new solutions are accepted based on probability quality. The pseudo algorithm is shown in Algorithm 1.

a. Cost calculation

The cost of a given state assignment is usually calculated by the number of literals in the Boolean expression that represents the FSM. But for comparison reasons, the factor that decides the quality is the area and the type of flip-flops used to build the sequential circuit described by the FSM. The equation is the same as [10]:

$$Area = P \times (2i + 3\log_2 N + o + n_{jk}) \tag{5}$$

where P are the number of product terms, N is the number of states, i is the number of inputs, o is the number of outputs, and n_{jk} is the number of flip-flops JK used in the physical form of the circuit. The JK flip-flops have a more complex structure than the other types of flip-flops, which translates to a larger usage of physical space. The cost calculation penalizes the use of JK flip-flops since it means an increase in the physical space of the resulting circuit. In this paper are used only type D flip-flops, thus we do not utilize the term n_{jk} in the cost calculation.

b. Minimization

One important step in the whole process, is the conversion of the FSMs given by the LGSynth89 benchmark suite [18]. The files are in .KISS2 format are converted to .PLA (Programmable Logic Array) format, then fed to the well-known ESPRESSO [19] Logic Minimizer, a program from the SIS Logic Synthesis System. ESPRESSO is required for calculating the objective function. That is, to obtain the Boolean

Algorithm 1: Pseudo-code of the *Simulated Annealing*

```
Result: Write the results in a plot and a CSV file
1  Select an initial solution;
2  Select a starting temperature;
3  Select a cooling factor;
4  Select a few loops in each temperature iteration;
5  while temperature not stabilized do
6      for repetition cycles do
7          Read the .kiss2 file;
8          Convert to a .pla file;
9          Get cost of the current solution with
             ESPRESSO;
10         Generates new solution;
11         Get cost of the new solution with ESPRESSO;
12         Calculates the energy variation using (4);
13         if new solution is better than current solution
             then
14             current solution ← new solution;
15         end
16         else
17             if probability allows then
18                 current solution ← new solution;
19             end
20         end
21         if current solution is better than global
             solution then
22             global solution ← current solution;
23         end
24     end
25     Apply cooling factor;
26 end
```

expressions that represent the FMS, as explained in Section II.

The ESPRESSO program is a C language implementation of the ESPRESSO algorithm [20, 21], that receives the PLA converted from the .KISS2 file outputs a .PLA format containing the minimized FSM, as well as the number of inputs, number of outputs, and the number of products terms, which represents the state transitions of the FSM, is used in the equation that calculates the cost (Eq. (5)). Conversion is just a form of binary and labeled representation of the FSM. For this reason, the details will be omitted, but details on representation can be obtained in the references.

IV. Results and Comparison

The SA starts with a temperature of 100, in the first half, the cooling factor is 1.2, in the second half the factor changes to 0.8, the same used in[10], slowing down the temperature curve. The initial temperature parameter was chosen based on the tests made, where all the FSM achieve stability by the end of the iterations. The algorithms (both our SA and the methods used in the comparison) were evaluated using the benchmark LGSynth'89 [18]. The benchmark LGSynth'89 is a set of examples with 41 FSM presented in the International Workshop on Logical Synthesis in 1989. The information about the number of states, size of the input, and output of the FSM are shown in Table 5.

To run the tests was used a notebook equipped with Intel i7 6th generation, 16 GB of RAM. The code was written in Python and used the ESPRESSO script, which is written in C language, as a sub-process to minimize the state assignment generated.

The experiment results are shown in Fig. 4. The results are shown normalized. Most cases continue to improve the solution towards the end of 100 iterations. Furthermore, the lines 6 to 21 in Algorithm 1 were repeated 100 times. With a few exceptions, such as `donfile` and `sand`, most of the cases had continued to improve all over the iterations. These cases (`donfile` and `shiftreg`) are small FSM, so there was not much improvement in the solution over the iterations.

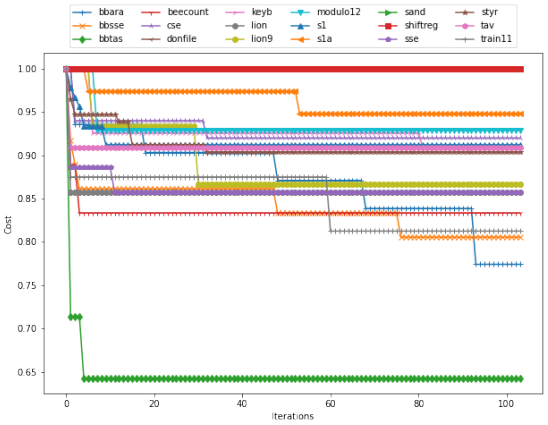


Fig. 4: Improved solution for all cases tested in the experiment compared to all iterations.

Table 6 shows the general results of the experiment carried out and a comparison with the work of [10, 22, 23]. For clarity of the results, all data from the other methods were extracted from the research of [10, 22, 23]. Only the second and tenth columns of Table 6 are the results of this experiment. Columns with the same method and values in parentheses mean that parameters were passed to the method to differentiate the search for the solution. The MUSTANG [2] (MUS for short) was run with `-p` and `-n` options, that correspond to fanout-oriented and fan in-oriented algorithms. The NOVA [24] was run with the `-e ig` option that causes the NOVA to be driven by input constraints and `-e ioh` option that causes it to be driven by both input and out constraints. The JEDI [25] was run with `-e o` option that uses the output dominant algorithm and `-e c` option that uses the coupled dominant algorithm.

The cases in which our results were better than or equal to all the compared methods are highlighted in bold. And it is highlighted in italic, all cases where the results are better than or equal to the methods compared except for the GESA (Guided Evolutionary Simulated Annealing) method. The GESA method is a hybrid algorithm (SA and Genetic Algorithm) available in [10]. The GESA is our focus to compare our experiment.

From these data, it is possible to notice that our results were not far from the compared methods. Compared to GESA, the worst case is `modulo12` where our solution was 59 % worse than GESA. However, the best case was `donfile` where our solution was 10 % better than GESA. On

average, our results were 11 % lower than GESA. It is difficult to measure whether these are acceptable percentages, considering the simplicity of our implementation compared to the GESA hybrid method, it seems reasonable to consider it.

The important point of our results is the processing time. In all cases, better processing times were obtained than the GESA method. For the worst case, the `tav` case, there was an improvement of 63 % of the time. And for the best case, case `s1a`, there was an improvement of 95 %. On average, there was an improvement of 86 % for all cases.

The time improvement was an expected factor, since our algorithm is much simpler than GESA, and it does not work on a certain population as a Genetic Algorithm. One point that could threaten the validity of our experiment is the research age of [10], which is a paper published in 2000. However, it is important to note that GESA was implemented with the C language, whereas in this work the Python language. Python, for being interpreted, is a slower language than the C language. Another point is the hardware that was used. In GESA a SPARCstation 20 station was used. SPARCstation 20 supports up to 4 CPUs, and in the paper, there is no information about which CPU is used. In our experiment, as already mentioned, the Intel i7 6th generation was used. Despite being very different generations, a low-performance personal computer with a high-performance server is being compared.

V. Conclusions

In this research, the problem of finding near-optimal state assignment in a finite state machine has been considered. Moreover, the work shows an evaluation of the solutions provided by a simulated annealing algorithm. Specifically, it has been provided an answer to the following question "how distant are the results with a much simpler and faster metaheuristic with less computational efforts (without a population of solutions)?" and our contribution is a solution to the assignment states in a finite state machine with the near-optimal solution with less computational effort, using Simulated Annealing.

The results have shown that it is possible to have acceptable losses in the quality of the solution with a considerably small amount of processing time, i. e. with less computational effort. For example, for the cases `bbsse` and `cse`, where there was a time gain of 82.93% and 84.52%, respectively, at a cost of loss of solution quality of 2.67% and 2.57%, respectively. Moreover, the case `beyb`, where there has been a time gain of 71.91% with the best of solution with 5.35% of loss quality solution. However, given the gap between the computers used, there must be a fairer comparison of performance. Because it is a much simpler algorithm, it could also be important to carry out an experiment calculating the real computational cost or effort, measured in power or joules per instruction.

References

[1] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, no. 3, pp. 269–285, 1985.

[2] S. Devadas, H.-K. Ma, A. Newton, and A. Sangiovanni-Vincentelli, "Mustang: state assignment of finite state machines targeting multi-level logic implementations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 12, pp. 1290–1300, 1988.

[3] P. Molitor, "A survey on wiring." *Elektronische Informationsverarbeitung und Kybernetik*, vol. 27, pp. 3–19, 01 1991.

[4] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[5] V. Fabera, V. Janes, and M. Janesova, "Automata construct with genetic algorithm," in *9th EUROMICRO Conference on Digital System Design (DSD'06)*, 2006, pp. 460–463.

[6] N. Niparnan and P. Chongstitvatana, "An improved genetic algorithm for the inference of finite state machine," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 7, 2002, pp. 5 pp. vol.7–.

[7] S. Amellal and B. Kaminska, "Functional synthesis of digital systems with tass," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 5, pp. 537–552, 1994.

[8] B. Mitra, S. Jha, and P. Choudhuri, "A simulated annealing based state assignment approach for control synthesis," in *[1991] Proceedings. Fourth CSI/IEEE International Symposium on VLSI Design*, 1991, pp. 45–50.

[9] G. Hasteer and P. Banerjee, "Simulated annealing based parallel state assignment of finite state machines," in *Proceedings Tenth International Conference on VLSI Design*, 1997, pp. 69–75.

[10] I. Ahmad, F. Ali, and R. Ul-Mustafa, "An integrated state assignment and flip-flop selection technique for fsm synthesis," *Microprocessors and Microsystems*, vol. 24, no. 3, pp. 141–152, 2000.

[11] T. Floyd, *Sistemas digitais: fundamentos e aplicações*. Rio de Janeiro: Bookman Editora, 2009.

[12] H. Taub, *Circuitos Digitais e Microprocessadores*. Rio de Janeiro: Editora McGraw-Hill, 1984.

[13] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *SCIENCE*, vol. 220, pp. 671–680, 1983.

[14] E.-G. Talbi, *Metaheuristics: From Design to Implementation*, 06 2009, vol. 74.

[15] N. Hansen and C. Barlow, "17 - plastic deformation of metals and alloys," in *Physical Metallurgy (Fifth Edition)*, fifth edition ed., D. E. Laughlin and K. Hono, Eds. Oxford: Elsevier, 2014, pp. 1681–1764. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444537706000174

[16] J. Dréo, P. Siarry, A. Pétrowski, and E. Taillard, *Metaheuristics for Hard Optimization*. Springer, 2003.

[17] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*. Springer, 2019.

[18] S. Yang, "Logic synthesis and optimization benchmarks," Tech. Rep., Dec. 1988. [Online]. Available: https://ddd.fit.cvut.cz/prj/Benchmarks/index.php?page=download

[19] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, *SIS: A System for Sequential Circuit Synthesis*. Electronics Research Laboratory,Department of Electrical Engineering and Computer Science University of California, Berkeley, CA 94720, 1992.

[20] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for pla optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 727–750, 1987.

[21] ——, "Espresso-mv: algorithms for multivalued logic minimization," *Memorandum No. UCB/ERL M86/65*, 1985.

[22] P. Yip and Y.-H. Pao, "A guided evolutionary computation technique as function optimizer," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 1994, pp. 628–633 vol.2.

TABLE 5: TABLE OF CHARACTERISTICS FOR BENCHMARKS USED

Example	No. of states	No. of inputs	No. of outputs
bbara	10	4	2
bbsse	16	7	7
bbtas	6	2	2
beecount	7	3	4
cse	16	7	7
donfile	24	2	1
keyb	19	7	2
lion	4	2	1
modulo12	12	1	1
s1	20	8	6
s1a	20	8	6
sand	32	11	9
shiftreg	8	1	1
sse	16	7	7
styr	30	9	10
tav	4	4	4
train11	11	2	1

TABLE 6: COMPARISON OF COSTS OBTAINED BY SA.

Case	**SA**	GESA	MUS-P	MUS-N	NOVA(-e ig)	NOVA(-e ioh)	JEDI(-e o)	JEDI(-e c)	Runtime GESA (s)	**Runtime SA (s)**
bbara	*509*	432*	550	572	550	572	616	594	286.92	63.72
bbsse	*990*	900*	1122	1089	990	1089	1122	1089	1091.73	73.82
bbtas	*124*	120*	195	150	180	165	165	195	164.16	48.39
beecount	276	198*	228	228	228	228	209	228	215.12	52.47
cse	*1518*	1480*	1485	1584	1485	1815	1947	1980	1685.96	109.71
donfile	**750**	840	980	1020	960	940	900	620*	1078.40	111.44
keyb	**1408***	1457	3317	1798	1705	3162	1798	1860	1750.82	240.97
lion	*66*	55*	77	77	66	88	88	77	122.12	46.17
modulo12	172	108*	195	195	180	180	165	180	350.64	51.18
s1	**2832**	2849	3108	3552	3219	2775*	3182	2923	1833.00	151.94
s1a	2605	2470	3182	2886	2960	2701	1813*	2479	2638.84	115.16
sand	4600	3901*	5060	5014	4692	4554	4876	4830	2256.48	277.05
shiftreg	**48***	48	48	72	96	48	132	96	198.45	47.65
sse	*974*	875*	1122	1089	990	1089	1188	1122	1060.20	73.66
styr	*4400*	3854*	5117	4945	4429	4558	4816	4644	1845.40	385.75
tav	*180*	162*	198	198	198	198	198	198	139.22	50.68
train11	200	147*	238	238	204	187	221	187	495.59	51.82

[1]Values in bold mean improvement in the solution. Values in italic mean a deterioration of the solution.
The value marked with a '*' is the best solution for given benchmark

[23] ——, "Combinatorial optimization with use of guided evolutionary simulated annealing," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 290–295, 1995.

[24] T. Villa and A. Sangiovanni-Vincentelli, "Nova: state assignment of finite state machines for optimal two-level logic implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 9, pp. 905–924, 1990.

[25] F. Buijs and T. Lengauer, "Synthesis of multi-level logic with one symbolic input," in *Proceedings of the Conference on European Design Automation*, ser. EURO-DAC '91. Washington, DC, USA: IEEE Computer Society Press, 1991, p. 60–64.