Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 11 (2017) 1726 – 1734

Procedia MANUFACTURING

27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy

Evolutionary Algorithms for Programming Pneumatic Sequential Circuit Controllers

Sajaysurya Ganesh[a], Saravana Kumar Gurunathan[b,*]

[a]*Former Intern, Indian Institute of Technology Madras, Chennai 600036, India*
[b]*Associate Professor, Indian Institute of Technology Madras, Chennai 600036, India*

Abstract

Sequential actuation of pneumatic cylinders is a common form of automation in small and medium scale industries. By changing such actuation sequences to suit the different products being processed, flexible automation can be economically realized. However, changing the actuation sequence involves manually reprogramming Programmable Logic Controllers (PLC), which consumes time and hinders the implementation of flexible automation. This paper presents a novel methodology to automatically program PLCs by evolving logic equations using Genetic Algorithm and Genetic Programming for the desired actuation sequence. Case studies have been presented to demonstrate the possibility of using the proposed methodology to reliably implement flexible automation.

Peer-review under responsibility of the scientific committee of the 27th International Conference on Flexible Automation and Intelligent Manufacturing

Keywords: Flexible Automation; Genetic Algorithms; Genetic Programming: Programmable Logic Controller; Pneumatics

1. Introduction

Pneumatics is a cheap source of power that is extensively used to operate fixed automation systems in small and medium scale industries. Such automation systems generally actuate pneumatic cylinders repeatedly in a fixed

* Corresponding author. Tel.: +91-44-2257-4736
E-mail address: gsaravana@iitm.ac.in

Peer-review under responsibility of the scientific committee of the 27th International Conference on Flexible Automation and Intelligent Manufacturing
doi:10.1016/j.promfg.2017.07.299

sequence and thereby perform the same operation on successive products in a process line. In early 1970s, techniques were developed to derive sequential logic equations [1] that can be used to construct sequential pneumatic cylinder actuation circuits. Subsequently, these techniques were refined [2, 3] and are still being used in the industries, particularly for manually generating logic equations for programming PLCs. Availability of PLCs can be leveraged to convert existing fixed automation systems into flexible automation systems as the PLC programs can be dynamically changed to suit changes in product and process requirements. However, the current manually performed techniques for deriving logic equations must be automated to quickly reprogram PLCs without much downtime.

Previous attempts to automate generation of logic equations include object oriented approaches [4] using C++, expert systems like PNEUMAES [5], Web based collaborative environments for simulating pneumatic circuits [6]. Nevertheless, most industries still rely on manual techniques [7] for designing circuits and programming PLCs.

Existing logic synthesizers use procedures that depend upon the pattern recognition capability of humans which becomes inefficient when adapted to a computer. These methods are either based on the Karnaugh Maps [8] or on Quine McCluskey Algorithm [9] which is functionally similar to Karnaugh mapping. The Quine McCluskey algorithm has its practical limits as it is NP-Complete; in other words, the runtime of the Quine-McCluskey algorithm grows exponentially with the input size [10]. Further, these methods are limited to Boolean operators like AND, OR, NOT for generating logic equations and cannot use operators like XOR, which are supported by PLCs.

This paper presents an alternative logic synthesizer that uses Evolutionary Algorithms for deriving logic equations for sequential actuation of pneumatic cylinders. Evolutionary computing methods like Genetic Algorithm (GA) have been successfully applied to various combinatorial optimization problems and have succeeded in obtaining good solutions for several such problems [11, 12]. They have been used to solve high level logic synthesis for VLSI design [13, 14] and reversible logic circuits [15]. Recently, GA has also been used for optimizing the number of logic gates in synchronous sequential circuits to reduce the circuit complexity [16].

The authors of this paper have previously published a method to automate pneumatic sequential circuit design using Genetic Algorithm [17]. This paper develops upon that and employs Genetic Programming for logic synthesis. The subsequent sections detail the complete methodology by which the logic equations and the corresponding PLC programs can be derived from a given actuation sequence of pneumatic cylinders.

Nomenclature

A, B, C, …	Pneumatic cylinders – Actuators
…, X, Y, Z	Auxiliary memory variables (AMV) of a PLC; relays in electro pneumatics
…, X1, Y1, Z1	Set signal sent to a (AMV/Relay)
…, X0, Y0, Z0	Reset signal sent to a (AMV/Relay)
A1, B1, C1, …	Extend actuation signal sent to cylinder with the name that precedes 1.
A0, B0, C0, …	Retract actuation signal sent to cylinder with the name that precedes 0.
a0, b0, c0, …	Signal from position sensors sensing the retraction of cylinder with the name that precedes 0. Their value is 1 when the cylinder remains in the completely retracted position and is 0 otherwise.
a1, b1, c1, …	Signal from position sensors sensing the extension of cylinder with the name that precedes 1. Their value is 1 when the cylinder remains in the completely extended position and is 0 otherwise.
…, $\bar{x}, \bar{y}, \bar{z}$	The outputs of normally closed contacts of relays. Each of their value is 1 when the corresponding relay is in RESET condition and is 0 when the corresponding relay is in SET condition.
…, x, y, z	The outputs of normally open contacts of relays. Each of their value is 1 when the corresponding relay is in SET condition and is 0 when the corresponding relay is in RESET condition.

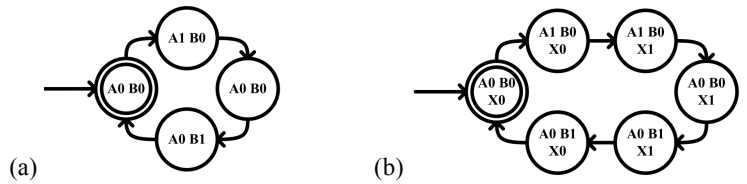


Fig. 1. (a) A1 A0 B1 B0 automaton; (b) A1 X1 A0 B1 X0 B0 automaton

2. Methodology

The proposed logic synthesizer has three major processing steps: modifying the input operation sequence to eliminate signal clash, converting the modified sequence into truth table and synthesizing logic equations from the truth table. The subsequent sections detail the process carried out in each step using the sample input sequence "A1 A0 B1 B0". This sequence encodes the operation of a punching press. The punch (cylinder A) extends (A1) and deforms the sheet metal blank and subsequently retracts (A0). The ejector (cylinder B) reciprocates (B1B0) to eject the deformed sheet. These cylinders have piston position sensors which sense the completion of an activity in the sequence so that the PLC can signal the commencement of the next activity in the sequence. The PLC program that control these actuators is essentially a set of logic equations that maps the state of position sensors to a particular actuation signal. So the operation sequence "A1 A0 B1 B0" can be expressed as an automaton in Fig. 1a. However, this automaton is invalid as the state (A0B0) leads to both (A1B0) and (A0B1). This kind of a signal clash (both cylinders A and B would receive extension and retraction signals simultaneously) is highly undesirable in sequential operations. It can be rectified by using an intermediate state (a virtual actuator X) and the sequence can be modified as "A1 X1 A0 B1 X0 B0". The corresponding modified automata is shown in Fig. 1b. This modification of the input sequence is the first step in logic synthesis and is carried out using Genetic Algorithm. It is advantageous to use truth tables instead of automaton diagrams for subsequent processing. The truth table for the unmodified operation sequence is shown in Table 1. It shows all possible states of position sensors. For each state (row), the next step in actuation (column) is marked by "1" – as that particular state should send an ON actuation signal. "a" and "b" are position sensors for cylinder A and cylinder B respectively. Physically there would be two position sensors for each cylinders. For example, in cylinder A, a0 is the position sensor sensing the retracted position of the piston, and a1 is the position sensor sensing the extended position of sensor. But, for the sake of simplicity, a0 and a1 have been replaced by "a" with two possible states (0 and 1) by assuming that a0 and a1 are mutually exclusive. This mutual exclusivity can be enforced while physically implementing the circuit using additional memory valves or by implementing relay latching techniques in case of electro pneumatics [17]. Again, Table 1 shows that the state (a=0, b=0) leads to simultaneous actuation of both A1 and B1, which would be addressed while modifying the sequence.

2.1. Evolving modified sequence using Genetic Algorithm

The Genetic Algorithm used for modifying the input sequence uses an evolutionary strategy (Fig. 2a) with $\mu + \lambda$ selection [11]. The genotype is shown in Fig. 2b. For most industrial applications, 4 auxiliary memory variables (W, X, Y, Z) are sufficient and hence there are 4 empty genes between genes representing the original sequence. The initial population (parents) of size μ is generated by randomly filling the empty genes with set/reset operations. The input sequence with no set/reset genes in-between is also included among parents. Offspring (of population size λ) are generated by applying mutation and crossover operators (Fig. 2b) over randomly selected parents. The genes that constitute the original sequence are kept silent (unchanging) and the mutation and crossover operations preserve the original sequence. This technique is based on the gene silencing operator described in [18]. Mutation replaces a randomly selected non-silent gene with either a random auxiliary memory operation or with an empty gene. As all the chromosomes in the population have the original sequence actuations at the same gene indices, the single point crossover with a common line of crossover across both parents preserve the original sequence as shown in Fig. 2b. To ensure diversity in population, out of the two offspring at the end of crossover, only one survives and each have an equal probability of survival. The parents for the next generation (best μ) are selected from the total population ($\mu + \lambda$) of parents and offspring of the current generation. Eq. 1 shows the inverse fitness function $f(x)$ for evaluating the chromosomes in a given population. The variables in Eq. 1 can be calculated from the truth table of the chromosome (modified sequence).

$$f(x) = \max(N_f) + (S_{ms} / \max(S_i)) - 1 \times C \tag{1}$$

Where,

N_f = Number of occupants in a field (row in corresponding truth table)
S_{ms} = Size of modified sequence (excluding empty cells)
S_i = Size of individual (including empty cells)
C = Boolean value (1 if the sequence is cyclic, 0 otherwise)

Table 1. Truth Table for A1 A0 B1 B0.

a	b	A1	A0	B1	B0
0	0	1		1	
0	1				1
1	0		1		
1	1				

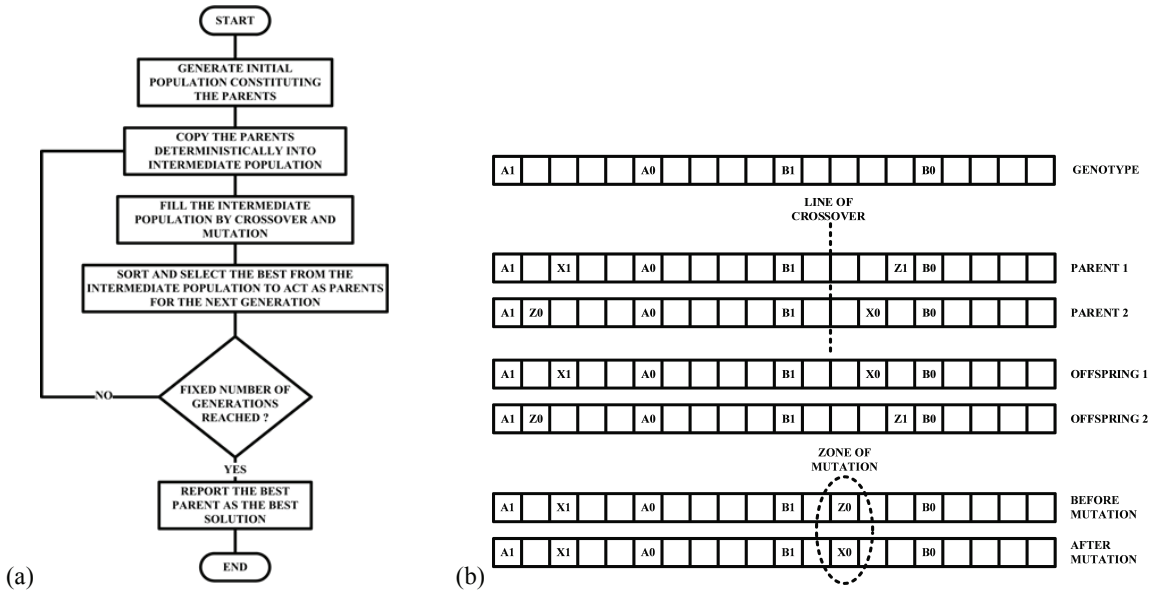


Fig. 2. (a) GA evolutionary strategy; (b) genetic operators

As chromosomes with minimum $f(x)$ is preferred during selection, the surviving chromosomes (modified sequence) at the end of evolution, would be short (with fewer auxiliary memory operations), valid (as the number of occupants in a row N_f would be at the minimum value 1 at which no state would lead to multiple actuations) and cyclic (the beginning state and the ending state would be identical, so that the sequence can be repeated). A valid chromosome to be used as a modified sequence must have $f(x) < 1$. The convergence of the algorithm is controlled by the parameters in Table 2. The crossover probability would be 1 – mutation probability. The values of these parameters (within the suggested range) for a given complexity (length of sequence and number of cylinders) of input sequence can be determined using response surface methodology [19]. The optimum modified sequence for the punching press input sequence is "A1 X1 A0 B1 X0 B0" and the corresponding truth table is shown in Table 3.

Table 2. Genetic Algorithm – Operation Parameters.

Parameter	Lower Limit	Upper Limit
Population Size μ	50	5000
Pool Multiplier $(\mu+\lambda)/\mu$	2	6
Mutation Probability	0%	100%

Table 3. Bare and filled truth table for A1 X1 A0 B1 X0 B0.

a	b	X	A1	A0	B1	B0	X1	X0	a	b	X	A1	A0	B1	B0	X1	X0
0	0	0	1						0	0	0	1	0	0	X	0	X
1	0	0					1		1	0	0	X	0	0	X	1	0
1	0	1		1					1	0	1	0	1	0	X	X	0
0	0	1			1				0	0	1	0	X	1	0	X	0
0	1	1						1	0	1	1	0	X	X	0	0	1
0	1	0				1			0	1	0	0	X	0	1	0	X
1	1	0							1	1	0	X	X	X	X	X	X
1	1	1							1	1	1	X	X	X	X	X	X

2.2. Filling the truth table

The next step is to fill the truth table of the modified sequence. Switching hazards can be eliminated by following the given rules for filling the table. The filled truth table for the sample modified sequence is shown in Table. 3.

- If a particular state (the bottom rows) is never visited by the sequence, fill the cells of that row with "x", which means that the value can either be "0" or "1". Essentially, it means that we need not care about the value.
- Locate cells with "1", and fill their complementing actuation signal in the same row with "0" so that an actuator never receives an extend and retract signal at the same time. If A1 has "1" then fill A0 in the same row with "0".
- To avoid inadvertent changes in the SET/RESET states of actuators or memory variables, in each column, identify a "0" and fill the cells underneath (cycle through top row) with "0" till a "1" is reached. Similarly, fill the cells underneath "1" with "x" till a "0" is reached. Fill remaining empty cells (if any) with "x".

2.3. Logic Synthesis using Genetic Programming

The final step is to synthesize logic equations that satisfy the filled truth tables. Each actuation (or set/reset AMV) operation in the truth table needs a logic equation to map various states to 1 (signal presence) or 0 (signal absence). The states that correspond to "x" need not be considered while synthesizing logic equations. Logic equations can be represented as a (Fig. 3b) binary tree, which can serve as genotype while evolving optimum logic equations using Genetic Programming. Again the evolutionary strategy (Fig. 3a) with $\mu + \lambda$ selection has been adopted.

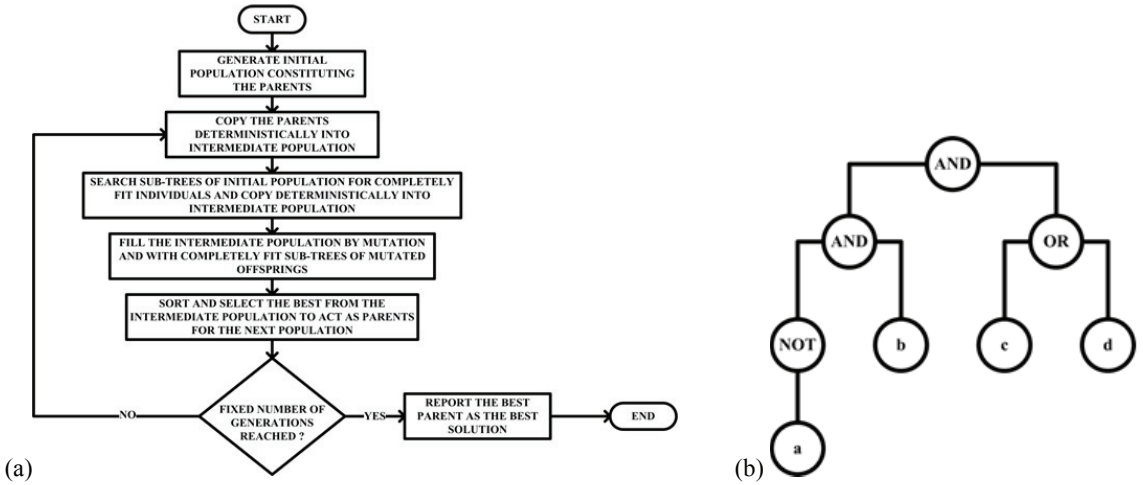


Fig. 3. (a) GP evolutionary strategy; (b) genotype representing ($\overline{a}$ AND b) AND (c OR d)

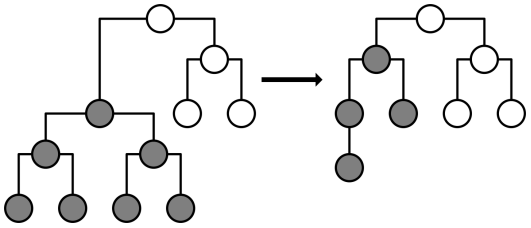


Fig. 4. sub tree mutation

The initial population (parents) of size μ is generated by using the Grow Method [20, 21]. The values of nodes are chosen randomly from the function set (AND, OR, NOT, XOR…) and the terminal leaves are selected from state variables (a, b, X…). The number of nodes in the randomly generated trees has been limited to 20, which is sufficient for most operation sequences. The offspring of population size λ is generated by the application of genetic operators over the parents. A portion of offspring population is generated by searching and extracting completely fit (maps all states to the actuation signal as per the truth table) sub-trees from all parents. As subtree crossover could disrupt some building blocks in logic equations [20], the crossover operator has been replaced by subtree search. Further [22] reports that headless-chicken crossover (semantically identical to subtree mutation) performs better than sub-tree crossover in this class of problems. Hence, the remaining vacancies in the offspring population is completely filled using sub-tree mutation operator. This is similar to evolution in bacteria, which reproduce asexually, where mutation is the only source of genetic variation. The authors found that a large population ($\mu = 500$), can save this crossover-less evolution from Muller's Ratchet [20, 23]. The sub tree mutation randomly selects a node in the parent tree and replaces the entire sub tree with a newly generated one. The depth of the tree undergoing mutation has been limited to 100. The sub tree mutation is described in Fig. 4. From this combined population of $\mu + \lambda$, the best μ individuals serve as the parents in the next generation. The following fitness function is used during selection.

$$f(x) = N_c + 1/N_i \tag{2}$$

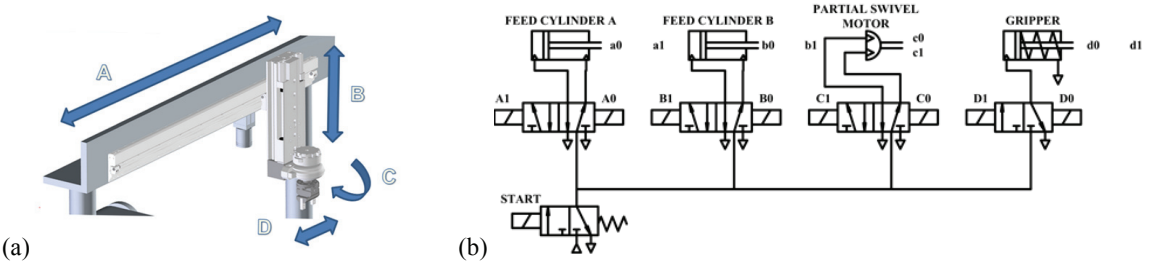
As the individuals with high fitness serve as parents for the next generation, the number of compatibilities with training set (N_c) would increase (making the resulting logic equation more valid) and the number of nodes in individuals (N_i) would decrease (resulting in a simple logic equation with minimum operators and states). The algorithm terminates after a given number of generations (determined based on problem complexity). The logic equations generated for the modified punching press operation sequence are A0 = x, A1 = $\overline{b \text{ OR } x}$, B0 = $\bar{x}$, B1 = $\bar{a}$ AND x, X0 = b, X1 = a. These logic equations can be instantly translated into PLC Instruction List program as per IEC 61131-1 by PLC Open technical committee [24, 25].

3. Case Study

A pick-and-place robot made of pneumatic actuators can ideally be used in a flexible automation setup. The robot under consideration (Fig. 5a) has A: cylinder for horizontal feed, B: cylinder for vertical feed, C: rotary actuator, D: gripper. While operating in the sequence "B1 D1 B0 C1 B1 D0 B0 C0", it can be used for re-orienting parts in a conveyer. The cylinder B extends (B1), the gripper actuates (D1) and holds the part, cylinder B retracts (B0), the part is re-oriented (C1), cylinder B extends again to place the part on the conveyer (B1), the part is released (D0) and finally actuators B and C return to their initial state (B0C0). Again, the same robot can be used for transferring defective parts from one conveyer to another, while operating in the sequence "B1 D1 B0 A1 D0 A0". The cylinder B extends (B1), the gripper holds the part (D1), the cylinder B retracts (B0), cylinder A positions the head over the sorting bin (A1), the part is released (D0) and the head finally returns to its initial position over the conveyer (A0). The proposed methodology can quickly calculate the necessary logic equations for programming the PLC and hence the program can be quickly changed while repurposing the robot. The pneumatic circuit is shown in Fig. 5b and the PLC program generated using the proposed methodology is shown in table 4.

Table 4. PLC Instruction Lists for the pick-and-place robot.

Program for sequence "B1 D1 B0 C1 B1 D0 B0 C0"	Program for sequence "B1 D1 B0 A1 D0 A0"
LD_BOOL c1 XOR(LD_BOOL d1) ST_BOOL b LD_BOOL d1 XOR(LD_BOOL c0) ST_BOOL bb LD_BOOL d1 OR(LD_BOOL b1) NOT ST_BOOL c LD_BOOL d1 AND(LD_BOOL b0) ST_BOOL cc LD_BOOL b1 AND(LD_BOOL c1) ST_BOOL d LD_BOOL c0 AND(LD_BOOL b1) ST_BOOL dd END	LD_BOOL d0 ST_BOOL a LD_BOOL b0 AND(LD_BOOL d1) ST_BOOL aa LD_BOOL d1 ST_BOOL b LD_BOOL d1 OR(LD_BOOL a1) NOT ST_BOOL bb LD_BOOL a1 ST_BOOL d LD_BOOL b1 ST_BOOL dd END

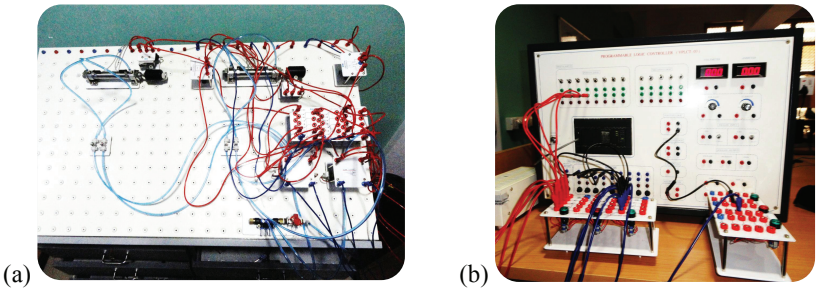


(a) (b)

Fig. 5. (a) pick-and-place robot; (b) electro-pneumatic circuit

Table 5. Logic synthesis performance comparison between Genetic Programming and Quine McCluskey

Operation Sequence	Modified Sequence	Genetic Programming		Quine McCluskey	
		# Nodes	#Rows	#Nodes	#Rows
A1 A0 B1 B0	A1 X1 A0 B1 X0 B0	4	6	4	8
A0 A1 C1 C0 B1 B0	A0 W1 A1 C1 X1 C0 W0 B1 X0 B0	10	10	11	32
A1 A0 A1 A0 A1 A0 B1 B0 C1 C0	A1 W1 A0 Y1 A1 Z1 A0 W0 A1 X0 A0 B1 Z0 B0 C1 X1 Y0 C0	24	18	32	128
		23	18	24	512
4. A1 A0 B1 B0 C1 C0 D1 D0 E1 E0 F1 F0	5. A1 Z0 A1 B1 W0 B0 C1 Y0 C0 D1 W1 D0 E1 Z1 E0 F1 Y1 F0				



(a) (b)

Fig. 6. (a) SMC® electro-pneumatic testbench; (b) GE_FANUC VersaMax® Micro PLC

Further case studies were performed with the input sequences shown in Table 5. The logic equations were evolved using a laptop PC equipped with Intel Core i7 3rd generation processor and 8GB DDR3 RAM, which took less than 10 seconds for generating the logic equations for each of the sequences listed in Table 5. The logic equations thus evolved were consistently simpler (less number of nodes) than those produced by Quine-McCluskey Algorithm.

The practical applicability of the proposed technique was verified in an SMC® electro-pneumatic testbench. The above mentioned laptop was connected to a GE-FANUC VersaMax® Micro PLC, which in turn was connected to the sensors and the actuators in the testbench. To change the operation sequence, the logic equations for the new sequence were evolved in the laptop and the resulting PLC programs were uploaded to the PLC, following which

the actuators could immediately start operating as per the new operation sequence, thus enabling flexibility in operation. A similar method can be used for the implementing the proposed technique in production lines. Based on preliminary time study with the testbench, a skilled operator could change the operating sequence of actuators in under 30 seconds.

6. Conclusion

A method to synthesize logic equations for programming PLCs that control sequential actuation of pneumatic cylinders has been proposed and detailed. The method uses a novel mix of Genetic Algorithm, Genetic Programming to convert a sequence of operations into its corresponding set of logic equations. Its capability in generating logic equations for different operating sequences of varying complexity has also been demonstrated through case studies. The proposed method can be used to quickly reprogram PLCs with dynamic changes in operational requirements and hence can be used to convert existing fixed-sequence automation systems into flexible automation systems. Although uncommon, some production lines operate actuators in a synchronized set of parallel actuation sequences. The current implementation cannot yet cater such complex needs, which could be the focus of future research in this domain.

References

[1] R.M.H. Cheng, K. Foster, Systematic method of designing fluidic – pneumatic control circuits, Proceedings of the Institution of Mechanical Engineers 1847-1982, 186 (1972) 401-408.
[2] C.H. Chung, K. Tam, G. Liu, Pneumatic sequential circuits, Industrial Engineering Journal, 1 (1980) 27-29.
[3] P. Rohner, Fluid power logic circuit design : analysis, design methods and worked examples / Peter Rohner, Macmillan, London :, 1979.
[4] P.K. Wong, T.P. Leung, C.W. Chuen, W.H. Chan, Object-oriented CAD for electro-pneumatic sequential circuit design in low cost automation, in: EEE Symposium on Emerging Technologies and Factory Automation, ETFA '94, 1994, pp. 278-284.
[5] S.-k. Sim, P.S.K. Chua, Symbolic pattern manipulation of Karnaugh-Veitch maps for pneumatic circuits, Artificial Intelligence in Engineering, 10 (1996) 71-83.
[6] C. Yen, W.-J. Li, A web-based computer-aided pneumatic circuit design software, Simulation Modelling Practice and Theory, 11 (2003) 285-295.
[7] M.S. Bayoumi, Novel Method for Designing a Sequential Logic Controller with Intermediate Stop of Actuators, International Journal of Computer and Information Technology, 03 (2014) 643-650.
[8] M. Karnaugh, The map method for synthesis of combinational logic circuits, Transactions AIEE, Commiin & Electron, 72 (1953) 593 - 598.
[9] E.J. McCluskey, Algebraic minimization and the design of two-terminal contact networks, Bell System Technical Journal, 35 (1956) 1417-1444.
[10] T.K. Jain, D.S. Kushwaha, A.K. Misra, Optimization of the Quine-McCluskey Method for the Minimization of the Boolean Expressions, in: Fourth International Conference on Autonomic and Autonomous Systems (ICAS), 2008, pp. 165-168.
[11] M. Gen, R. Cheng, Genetic algorithms and engineering optimization, Wiley, New York, 2000.
[12] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., 1989.
[13] K. Ohmori, P. Eklund, J. Daalder, Simultaneous Scheduling and Allocation in High-Level Synthesis using a Genetic- Algorithm, IEEE International Conference on Computer-Aided design (CADEX '96), (1996) 118-127.
[14] K. Ohmori, T. Kasai, Logic synthesis using a genetic algorithm, in: IEEE International Conference on Intelligent Processing Systems, ICIPS '97, 1997, pp. 137-142 vol.131.
[15] M. Lukac, M. Pivtoraiko, A. Mischenko, M. Perkowski, Automated Synthesis of Generalized Reversible Cascades using Genetic Algorithms, In Proceedings of Fifth International Workshop on Boolean Problems, (2002) 33-45.
[16] T. Yanyun, C. Jian, Z. Yuzhen, L. Jiajun, L. Minglu, Using module-level Evolvable Hardware approach in design of sequential logic circuits, in: IEEE Congress on Evolutionary Computation (CEC), 2012, pp. 1-8.
[17] G. Sajaysurya, G.S. Kumar, Hybrid intelligent systems for pneumatic sequential circuit design: Hybridizing genetic algorithms and rule-based logic programming, in: 11th International Conference on Hybrid Intelligent Systems (HIS), 2011, pp. 241-246.
[18] S. Siva Sathya, S. Kuppuswami, Gene silencing—A genetic operator for constrained optimization, Applied Soft Computing, 11 (2011) 5801-5808.
[19] R.H. Myers, Response surface methodology, Allyn and Bacon, Boston, 1971.
[20] W. Banzhaf, P. Nordin, R. Keller, F. Francone, Genetic Programming : An Introduction On the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann Publishers, 1997.
[21] J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems), MIT Press, 1992.
[22] P. Angeline, Subtree Crossover: Building Block Engine or MacroMutation?, in: K.D. J. Koza, M. Dorigo, D. Fogel, M. Garzon, H. Iba and R. Riolo (Ed.) Genetic Programming 1997: Proceedings of the Second Annual Conference, Morgan Kaufmann, San Francisco, CA, 1997, pp. 9-17.
[23] H.J. Muller, Some Genetic Aspects of Sex, The American Naturalist, 66 (1932) 118-138.
[24]K.-H. John, M. Tiegelkamp, IEC 61131-3: programming industrial automation systems : concepts and programming languages, requirements

for programming systems, decision-making aids, Springer, Berlin, 2010.
[25] PLCOpen, Technical paper—XMLformats for IEC61131-3—Version 2.01, (2009).