Action Scheduling Optimization using Cartesian Genetic Programming

Marco André Abud Kappel
Centro Federal de Educação Tecnológica Celso Suckow de Fonseca – CEFET/RJ
Nova Friburgo, Brazil
marco.kappel@cefet-rj.br

***Abstract*—Action scheduling optimization is a problem that involves chronologically organizing a set of actions, jobs or commands in order to accomplish a pre-established goal. This kind of problem can be found in a number of areas, such as production planning, delivery logistic organization, robot movement planning and behavior programming for intelligent agents in games. Despite being a recurrent problem, selecting the appropriate time and order to execute each task is not trivial, and typically involves highly complex techniques. The main objective of this work is to provide a simple alternative to tackle the action scheduling problem, by using Cartesian Genetic Programming as an approach. The proposed solution involves the application of two simple main steps: defining the set of available actions and specifying an objective function to be optimized. Then, by the means of the evolutionary algorithm, an automatically generated schedule will be revealed as the most fitting to the goal. The effectiveness of this methodology was tested by performing an action schedule optimization on two different problems involving virtual agents walking in a simulated environment. In both cases, results showed that, throughout the evolutionary process, the simulated agents naturally chose the most efficient sequential and parallel combination of actions to reach greater distances. The use of evolutionary adaptive metaheuristics such as Cartesian Genetic Programming allows the identification of the best possible schedule of actions to solve a problem.**

***Index Terms*—Cartesian Genetic Programming, Action Scheduling, Evolutionary Computation, Optimization**

I. Introduction

Action scheduling optimization is a problem that involves chronologically organizing a set of actions, jobs or commands in order to accomplish a pre-established goal. This kind of problem can be found in a number of areas, such as production scheduling [1], delivery logistic organization [2], robot motion planning [3], behavior programming for intelligent agents in games [4], [5], and even synthesizing feasible sequences of movements for a mobile arm-equipped robot for space exploration [6]. A classic example of a scheduling problem is the Job-Shop Scheduling Problem [7]. The objective of this problem is to schedule the execution of multiple jobs on several machines. There are many variations of this problem, each one defining different constraints, such as a limited number of machines that can operate simultaneously, or the specification of a dependency order for each job. In addition, each variation defines a main objective, such as to minimize the length of the schedule or to optimize the total number of performed jobs. Recent instances of the action scheduling optimization problem seek to define the best sequence of commands that must be passed to a robot so it performs some complex operations. For example, what should be the rotations to be applied to robot joints, and at what times, to make it walk great distances with a natural motion [8], [9], or to make its mechanical arm to move an object to a different location [10]. Despite being a recurrent problem, developing an algorithm to select the appropriate time and order to execute each task is not trivial, and typically involves highly complex techniques, as Dynamic Programming. The main objective of this work is to provide a simple alternative to tackle the action scheduling problem, by using Cartesian Genetic Programming as an approach.

II. Problem Description

A. Action Scheduling Optimization

Scheduling problems are defined by the need to organize various tasks in a particular order in a timeline. The schedule must optimize some specified value, such as total elapsed time, total energy cost or total machine idle time, while satisfying a set of pre-defined constraints. In general, the space of possible solutions for most real problems is far too large to brute-force or undirected search methods be feasibly applied. These characteristics make them NP-complete problems [7]. This means there is no known deterministic algorithm for an optimal solution. The first proposed solutions to these problems adopted the Dynamic Programming search method [7], but the combinatorial nature of most scheduling problems encourages the use of meta-heuristics as Simulated Annealing [11], Tabu Search [12] or Bottleneck Heuristics methods [13]. In addition, schedule optimization has become a major application field for evolutionary computation [14]. For example, Genetic Algorithms (GA) has been applied to task planning optimization [15] and multi-criteria scheduling optimization [1], while Genetic Programming (GP) has been applied in the Job-Shop Scheduling Problem [16] and in the classic one-machine scheduling problem [17].

B. Intelligent Agents Action Control

The goal of an intelligent agent is to build an action script that must be executed in order to accomplish an objective task. Thus, this problem can also be seen as a scheduling problem. For example, a possible action is to command a virtual robot to rotate the right elbow clockwise. In this case, an action can be seen as the rotation of one joint of the agent, clockwise

2643-6264/19/$31.00 ©2019 IEEE
DOI 10.1109/BRACIS.2019.00059

or counter-clockwise. Another action is to rotate the left knee counter-clockwise. Every action must be performed in a fixed number of time steps. In the present work, each time step represents one second. The time step can vary depending on the granularity and precision needed to complete the task. The goal is to define how long and at what time each action must be performed by the robot in order for it to travel greater distances. The solution to the problem is an optimal script of actions that contains the timetable of actions the robot must perform in order to complete the task. The present paper studies two simplified action control scheduling test cases involving simulated robot agents. In order to make possible to control all external factors and to test only the effects of the application of the proposed technique, both robots were created and simulated in a bi-dimensional virtual environment. Fig. 1 shows both robots, for case I and case II, respectively.

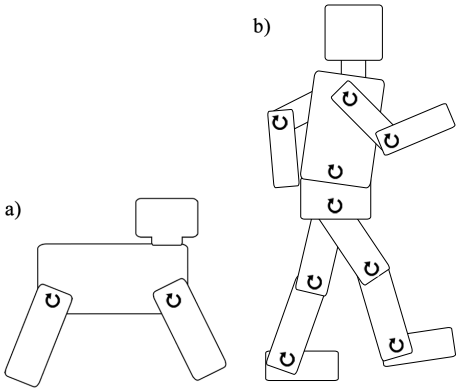

Fig. 1. Virtual simulation of two bi-dimensional robots. a) Case I: simple robot with two legs and two rotational joints. b) Case II: humanoid robot with eleven rotational joints.

The first test case involves a simple virtual robot having only two joints, one for each leg (Fig. 1a). In this case, the only possible actions the robot can perform are to rotate each joint clockwise or counter-clockwise, making four possible actions in total. The second test case involves a humanoid robot having eleven joints: one for each shoulder, one for each elbow, one for the lumbar, one for each leg, one for each knee and one for each ankle, making 22 possible actions in total (Fig. 1b). The goal for both instances of this problem is to determine the script of actions the robots must execute in order to walk as far as possible. In other words, the optimization problem is to maximize the travelled distance of each virtual robot.

III. Methodology

Manually inputting a sequence of action commands into a robot, or into a game character, is an ordinary task. Conversely, to identify an optimum script of action commands that will make the agent to achieve certain goal is a real challenge. The present paper proposes the application of a simple and efficient modeling strategy of Cartesian Genetic Programming.

A. Cartesian Genetic Programming

Genetic Programming (GP), an extension of Genetic Algorithms, is an evolutionary computation approach, originally developed by Koza [18], and also relying on the Darwinian principle of natural selection. Genetic Programming submits a population containing a number of individuals to an evolutionary process. These individuals are randomly generated candidate programs in an iterative process that, over the course of generations, selects promising individuals for procreation and the formation of new offspring through genetic operators. To model a problem by using Genetic Programming (GP), the most important step is to find the best individual representation of a solution candidate. In an action scheduling problem, each individual of a GP population can represent a different action script, describing which action must be performed in a timeframe. Throughout the generations, the action scripts will evolve until the best-fitted individual emerges. In order whether to apply the classic Genetic Programming or its Cartesian variation, it is essential to specify two critical factors: a function set and a terminal set. These sets establish the basic instructions that, when combined for an individual, express a solution candidate.

The function set establish the collection of available operators for each individual to use. The output of each operation must be compatible as operand to any other operator in the function set. This makes possible to randomly generate individuals by combining these elements in different ways. The elements of the terminal set are the basic operands available for the program. These operands can act as entries to the problem, and must be compatible to any operators described in the function set. The initial individuals of a GP population are simple random combinations of the elements of function set and terminal set. For GP to be successful, it suffices that the solution to the problem could be expressed in the form of a combination of elements of the function set and terminal set. If enough terminals or operators are not made available, the solution may never be found. On the other hand, if the elements are sufficient, the method will have the potential to generate solutions that are competitive with those of human specialists [19].

In the present work, each individual represents a different action schedule. Thus, all possible actions each agent can perform were chosen as terminals. Each joint of the robots can be rotated clockwise or counter clockwise, originating two actions per joint. For simplification purposes, the angular velocity of each rotation was fixed and constant throughout all simulations. Table 1 shows the elements of the terminal set.

The elements of function set must be operators that can receive as entries and combine the direct actions described by terminals or the result of another combination of actions. Thus, to make the construction of a wide variety of action schedules possible, the elements chosen to compose the function set denotes the following operations: sequential execution and parallel execution, as shown in Table 2. These functions accept only two entries, which can be another function or an action

TABLE I
TERMINAL SET

Test Case	Joint	Available Terminals	
		Clockwise Action Symbol	*Counter Clockwise Action Symbol*
I	Left leg	LLC	LLCC
	Right leg	RLC	RLCC
II	Left Elbow	LEC	LECC
	Right Elbow	REC	RECC
	Left Shoulder	LSC	LSCC
	Right Shoulder	RSC	RSCC
	Lumbar	LC	LCC
	Left leg	LLC	LLCC
	Right leg	RLC	RLCC
	Left Knee	LKC	LKCC
	Right Knee	RKC	RKCC
	Left Ankle	LAC	LACC
	Right Ankle	RAC	RACC

TABLE II
FUNCTION SET

	Sequential execution	*Parallel execution*
Description	Execution of one action starts only after the finalization of the other.	Execution of one action starts simultaneously with the other.
Symbol	– –	//

of the terminal set. When activated, an action will be executed for exactly one time step. In the present work, each time step corresponds to 60 frames of the simulation, which translates to a second.

The main difference between classic GP and CGP is the individual representation. In traditional GP, each individual is a direct implementation of an expression tree [18]. CGP, on the other hand, represents each individual in form of an acyclic directed graph, as a two-dimensional grid of computational nodes [20]. Because of its natural graph structure, CGP has been successfully applied in the design of logical [21] and electrical [22] circuit's optimal topology.

Each computational node has a number of genes, which represents one between three things: an input, the operator that will be used on the inputs, or a node output. Fig. 2 is an example of a particular node, with two inputs. The CGP genotype is composed by node columns. The first column corresponds to terminals.

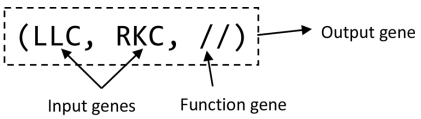

Fig. 2. Example of CGP node: a simultaneous execution of left leg clockwise rotation with right knee clockwise rotation.

All following columns of the individual grid are formed by common CGP nodes (Fig. 2). These nodes are direct applications of an operator from the function set on nodes of prior columns. The CGP graph is directed and feed-forward, i.e., a node can only have its inputs connected to a node from a column on its left. The output of a complete individual is taken at the output of the first node on the last column. The phenotype of a CGP genotype is the actual action schedule it represents. The decoded form of the genotype can be illustrated as the correspondent expression tree, dynamically generated during the method execution. Fig. 3 shows an example of the genotype and phenotype of a complete CGP individual for the problem under study. In this example, the CGP grid was configured with 2 lines, 4 columns and only 2 actions on the terminal set.

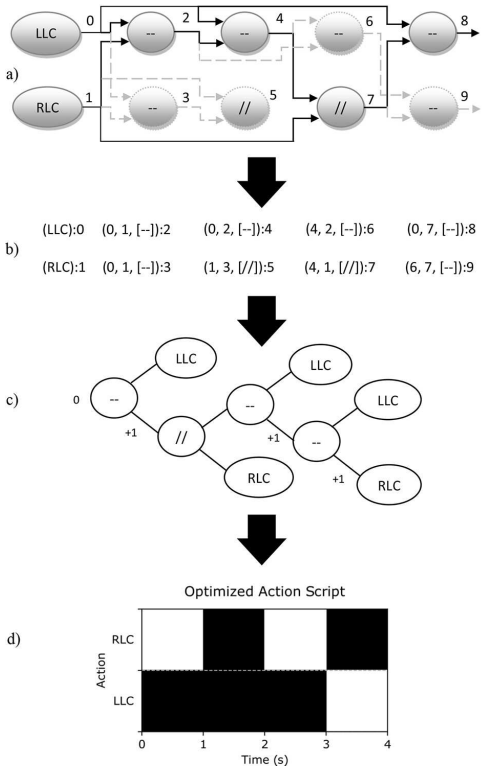

Fig. 3. Complete genotype and phenotype of a CGP individual representing an action schedule: a) acyclic directed graph genotype; b) numerical representation; c) expression tree representation; d) individual phenotype.

Lastly, it is necessary to define the fitness function that will evaluate the individuals. The fitness function considered in the present work is the sum of the total distance traveled by the head of the robot with the final height of it, as described in (1). No other objectives were given to the individual, so each virtual agent will try to find the most efficient way to walk great distances while maintaining its head up.

$$fitness = robot\_head_x + robot\_head_y \quad (1)$$

The CGP algorithm is similar to any other evolution-based optimization method, as described in [20]. First, an initial population of randomly generated individuals is created. Then, genetic operators are applied on each individual, generating offspring. In the present paper, the following genetic operators

were applied: mutation, crossover and selection. Mutation and selection works as described in the original formulation of CGP [20]. On the other hand, the crossover technique was not originally part of the CGP method. There are many variations of crossover implementations for CGP, but the one described in [23] was chosen for the present work, due to its good results in comparison with others and low impact on the original CGP structure.

The best individuals between old and new ones are selected to form a new generation of individuals. This process continues until a predetermined maximum number of generations is met. One of the greatest advantages of CGP is the presence of nodes that can be ignored in the decoding of the genotype into phenotype. For example, in Fig. 3, nodes 3, 5, 6 and 9 are completely inactive on the phenotype. This phenomenon is often called neutrality, and has been shown to be extremely beneficial to the evolutionary process [24], [25]. The description of CGP configuration parameters and the adopted values for each case are shown in Table 3.

Those values were selected empirically, after several test runs, and considering the suggestions of Miller et al. [21]. The most important settings are those involving the graph structure. If enough nodes are not available, the process might never achieve the required result. On the other hand, an excess of available nodes can cause evolution to take more time than necessary to optimize the objective function.

TABLE III
CGP CONFIGURATION PARAMETERS VALUES AND DESCRIPTION.

Parameter	**Description**	**Value**	
		Case I	***Case II***
Population size	Total number of individuals in a population.	40	100
Mutation rate	Percentage of genes that will mutate.	0.02	0.02
Selection rate	Percentage of the best fitness that is necessary to survive to the next generation.	0.7	0.7
Crossover rate	Percentage of population that will have genes crossed over.	0.2	0.2
nº of rows	Number of rows in the graph structure.	2	2
nº of columns	Number of columns in the graph structure.	20	100
Levels back	Indicates how many levels of previous columns a node can be connected to.	5	50
Function set size	Number of operations in the function set.	2	2
Terminal set size	Number of terminals in the terminal set.	4	22
Max. generations	Limit number of evolution generations.	100	300

Every fitness evaluation of an individual corresponds to an execution of a complete action schedule in a simulated environment. Therefore, a set of constraints were also stipulated for the simulations. First, the maximum elapsed time the virtual robots have to execute the designated action script was of 30 seconds. In addition, if the head of the robots gets below a fixed limit of 80% of the robot's total height, the individual execution is terminated and the fitness value is measured at that point. These constraints makes the evolutionary process faster and enhances the probability that the action schedule describes a more natural stand-up walking movement.

IV. RESULTS AND DISCUSSION

The effectiveness of the proposed methodology was tested by performing an optimization on two different action scheduling problems of virtual walking agents, running in a simulated environment. For each case, 10 repetition runs of the CGP application were executed. Table IV shows the statistical results for these runs. Fig. 4 and Fig. 5 shows the convergence evolution for cases I and II, respectively.

TABLE IV
STATISTICAL RESULTS FOR 10 RUNS OF CASE I AND II.

	Case I	***Case II***
Best Fitness	13.9930	11.2981
Worst fitness	10.3856	7.3151
Mean	12.5388	9.0577
σ	1.0502	1.1190

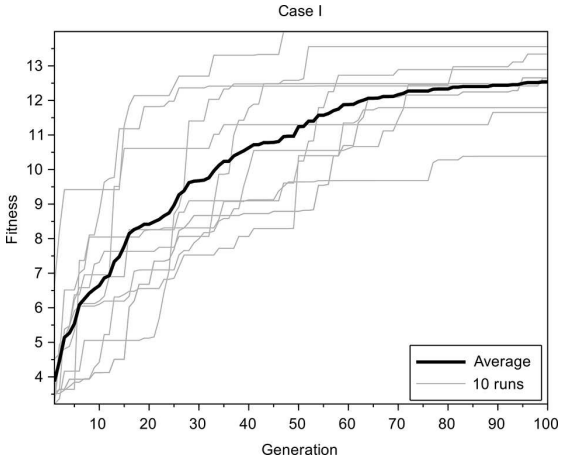

Fig. 4. Convergence evolution for case I.

The phenotypes of the best solution from each case are presented in Fig. 6 and Fig. 7, in the form of an action schedule for the respective simulated robot. Fig. 7 shows CGP was able to identify an action pattern that is efficient to make the virtual robot to steadily walk forward, for case I. The same action pattern was performed repeatedly, with the only exception being at 27 seconds of execution. Possibly, if more generations were made available, the action scheduling would repeat the pattern also in that moment. On the other hand, no obvious pattern was identified for case II. As it is a much more complex case, having 22 degrees of freedom, further exploration of the CGP configuration parameters might be needed. Some possible exploratory strategies would be to increase the population size and the maximum number of generations. Either way, the

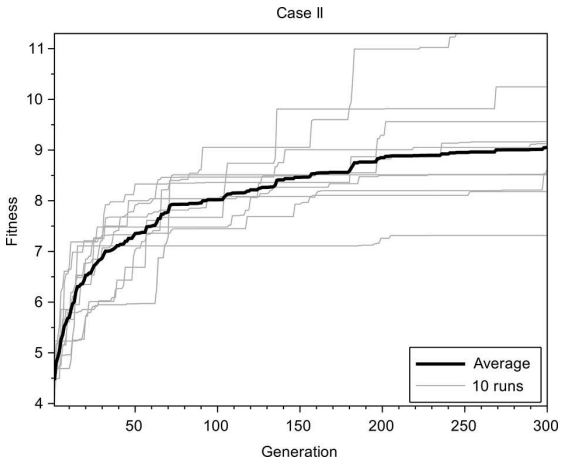

Fig. 5. Convergence evolution for case II

results for case II are satisfactory, since the virtual human-like agent was able to make complex simultaneous movements that enabled it to stand upright and move a few steps forward.

At the end of the evolutionary process, the virtual agent executes the best obtained action script. As this is an NP-complete problem, there might be no optimal solution. The obtained script represents only a sub-optimal solution. Furthermore, there is no known feasible deterministic way to find the best script. A subtle movement of one arm can collaborate for a small advance forward, or influence the entire balance of the system to cause a total collapse.

The best trajectories traversed by the head of each robot at the end of theirs evolutionary processes can be seen in Fig. 8 and Fig. 9, for cases I and II, respectively. In both cases, results showed the simulated robots naturally chose the most efficient sequential combination of actions to reach greater distances, improving on each generation of the evolutionary process.

V. Conclusions

The use of evolutionary adaptive metaheuristics such as Cartesian Genetic Programming on action scheduling problems allows the identification of the best possible combination of action sequences to achieve a goal. Through a series of evolutionary generations, only the best scripts were able to survive and evolve into a potential solution. At the end of the evolutionary process, ingenious combinations of actions were revealed as the most effective way to make each agent move longer distances forward while keeping his head in an elevated height.

On the present work, the fitness function being optimized described, as a single goal, to walk forward as far as possible, maintaining the head up. The same CGP methodology can be applied on other practical cases, with any other constraints or goals, like climbing up stairs or avoiding obstacles. After a simulated evolution, CGP generates the best possible action script and the agent would only have to follow the commands to achieve the goal. As this is an NP-complete problem, even the best solution found is only a sub-optimal solution. Therefore, further investigations are needed to explore the limits of this strategy.

References

[1] M. Kljaji, I. Bernik, and U. Breskvar, "Production planning using simulation and genetic algorithms in multi-criteria scheduling optimization." Frankfurt am Main: P. Lang verlag, 2003, pp. 193–208.

[2] C. A. Silva, T. A. Runkler, J. M. Sousa, and J. S. da Costa, "Optimization of logistic processes in supply-chains using meta-heuristics," *Lect Notes Comput Sc*, vol. 2902, pp. 1232–1238, 2003.

[3] G. Capi, Y. Nasu, L. Barolli, K. Mitobe, and K. Takeda, "Application of genetic algorithms for biped robot gait synthesis optimization during walking and going up-stairs," *Adv Robotics*, vol. 15:6, pp. 675–694, 2001.

[4] A. Nareyek, "Review: Intelligent agents for computer games," in *Computer and Games - Second International Conference*, vol. 2063. Berlin, Heidelberg: Springer, 2001, pp. 414–422.

[5] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artif Intell*, vol. 90, pp. 281–300, 1997.

[6] D. Díaz, M. D. R-Moreno, A. Cesta, A. Oddi, and R. Rasconi, "Appying ai action scheduling to esa's space robotics," *11th Symposium on Advanced Space Technologies in Robotics and Automation Proceedings*, pp. 5B_4–8, 2011.

[7] J. J. van Hoorn, A. Nogueira, I. Ojea, and J. A. Gromicho, "Solving the job-shop scheduling problem optimally by dynamic programming," *Comput Oper Res*, vol. 78, p. 281, 2017.

[8] M. Sadedel and A. Yousefi-koma, "Offline path planning, dynamic modeling and gait optimization of a 2d humanoid robot," Tehran, Iran, 2014.

[9] T. Arakawa and T. Fukuda, "Natural motion trajectory generation of biped locomotion robot using genetic algorithm through energy optimization," in *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No.96CH35929)*, vol. 2, Oct 1996, pp. 1495–1500 vol.2.

[10] D. Dey, T. Y. Liu, M. Hebert, and J. A. Bagnell, "Contextual sequence prediction with application to control library optimization," in *Robotics: Science and Systems 2012*, Sydney, Australia, Jul 2012.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1985.

[12] F. Glovar, "Tabu search: a tutorial," *Interfaces*, vol. 20(3), pp. 74–94, 1990.

[13] L. Greenwald and T. Dean, "Monte carlo simulation and bottleneck-centered heuristics for time-critical scheduling in stochastic domains." Tucson, Arizona: Elsevier Science, 1994, pp. 179–190.

[14] C. Dimopoulos and A. M. S. Zalzala, "Recent developments in evolutionary computation for manufactoring optimization: problems, solutions and comparisons," *IEEE Trans Evol Comput*, vol. 4(2), pp. 93–113, 2000.

[15] W. Jakob, M. Gomes-Schleuter, and C. Blume, "Application of genetic algorithms to task planning and learning," in *Conference: Parallel Problem Solving from Nature 2 - PPSN-II*, Brussels, Belgium, 1992, pp. 28–30.

[16] K. Miyashita, "Job-shop scheduling with genetic programming," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, Las Vegas, Nevada, 2000.

[17] C. Dimopoulos and A. M. S. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Adv Eng Softw*, vol. 32, pp. 489–498, 2001.

[18] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[19] J. R. Koza, F. B. III, D. Andre, and M. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[20] J. F. Miller, *Cartesian Genetic Programming. Natural Computing Series*. Berlin, Heidelberg: Springer, 2011.

[21] J. F. Miller, P. Thomson, and T. Fogarty, "Designing electronic circuits using evolutionary algorithms: Arithmetic circuits: A case study," pp. 105–131, 1998.

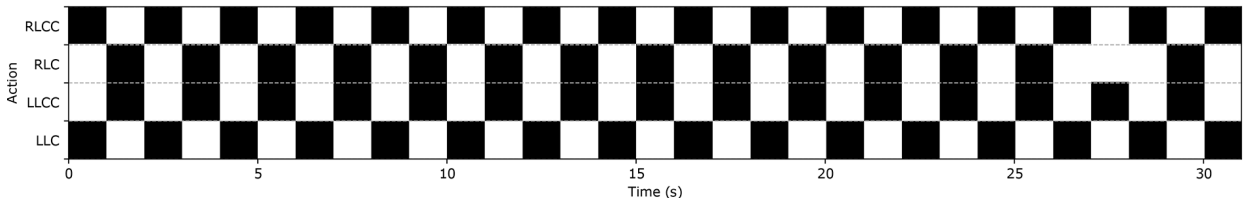

Fig. 6. Phenotype of the best individual at the last generation for case I, in form of an action schedule.

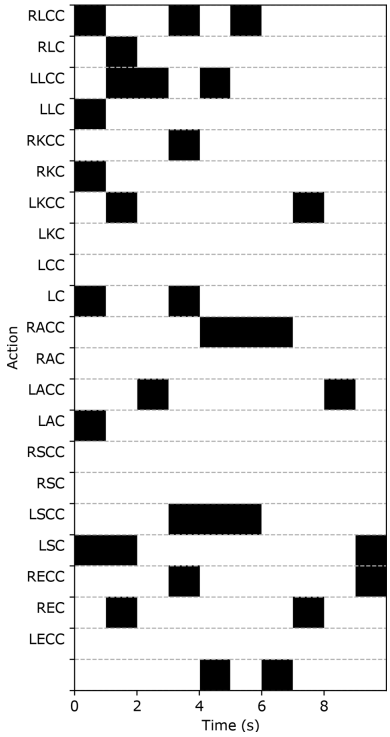

Fig. 7. Phenotype of the best individual at the last generation for case II, in form of an action schedule.

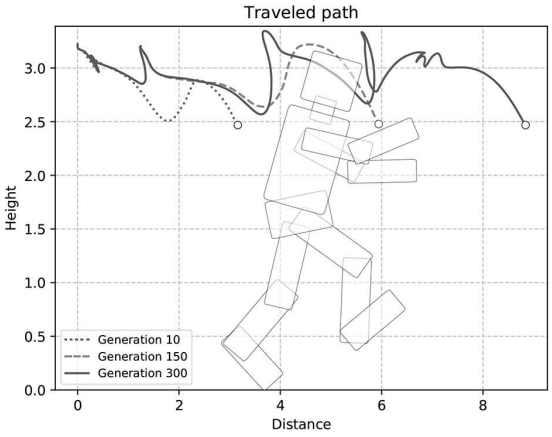

Fig. 9. Traveled path of the best individual in generations 10, 150 and 300, on the best run of case II.

[22] M. A. A. Kappel, R. P. Domingos, and I. N. Bastos, "Cartesian genetic programing applied to equivalent electric circuit identification," in *EngOpt 2018 Proceedings of the 6th International Conference on Engineering Optimization*. Springer, 2019, pp. 913–925.

[23] R. Kalkreuth, G. Rudolph, and A. Droschinsky, "A new subgraph crossover for cartesian genetic programming," *Genetic Programming*, p. 294–310, 2017.

[24] T. Yu and J. Miller, "Neutrality and the evolvability of boolean function landscape," in *Proc. European Conference on Genetic Programming*, vol. 2038. Springer, 2001, p. 204–217.

[25] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in cartesian genetic programming," *IEEE Trans. Evol. Comput.*, pp. 167–174, 2006.

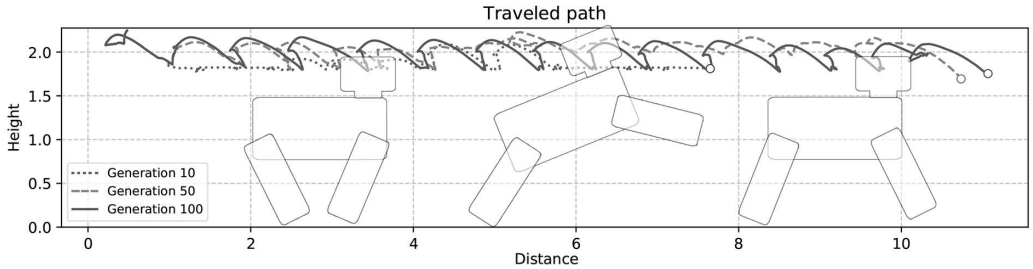

Fig. 8. Traveled path of the best individual in generations 10, 50 and 100, on the best run of case I.