Ant Colony Optimization

Brian Kelein Ngwoh Visas
Electronics Engineering
Hamm-Lippstadt University of Applied Sciences
Lippstadt, Germany
brian-kelein-ngwoh.visas@stud.hshl.de

***Abstract*—Ant Colony Optimization (ACO) algorithm is a stochastic algorithm used to solve the combinational optimization problem. Ant, The motivation of this algorithm comes from the unique behaviour of Ants living together in a colony. The ant colony walks along with the density of pheromone from the ant's nest to feeding sources. It leads to create the shortest path from the ant's nest to feeding sources. This Paper looks at the possibility of applying ACO to High-Level Synthesis. How we design a Fuzzy controller [1] with the help of ACO such that resources are appropriately optimized. Also, the design of the HACO-F algorithm [2], which is an algorithm that accelerates the ACO based on High-Level Synthesis on FPGA(Field Programmable Gate Array).which aims at Optimizing Data, reduce resource utilization and energy consumption, which in turn reduces computation time. We see that this algorithm tends to produce better results while solving an optimization problem.**

***Index Terms*—Ant Colony Optimization, Swarm Intelligence, High Level Synthesis**

I. Introduction

A. Definition of the Algorithm

Dorigo M in [3] introduced Ant Colony Optimization Algorithm in 1991. The motivation behind this algorithm is the behaviour of the Ant Colony. It has been used to solve several Optimization problems, such as the Travelling salesman problem(TSP) [4], network routing and vehicle routing problem and many more. Ant Colony Optimization is a procedure whereby the algorithm searches for the shortest path from an ants nests to the food source. Most of the Ants, while searching for food, walk along a path, and in the course of walking, pheromones are being deposited and with frequent movement along a particular path, the density of pheromone increases. Most of the time, it is the shortest path with the highest pheromone density, and the longest path has a lower pheromone density. This is how the Ants can find the shortest path to their source of food.

B. Swarm Intelligence and Biological background

The collective behaviour and social movements of a group of creatures (like bees, birds, fishes, ants, wasps, and many more.) are very significant, but the personal behaviour of a single creature often is uncomplicated. Because of the impressive applicability of the natural swarm systems, researchers have done numerous studies on such behaviours of social creatures. These systems proposed based on Artificial Intelligence (AI) concepts, in the late-80s, by computer scientists. In 1989, the first time, the term "Swarm Intelligence" was applied by G. Beni and J. Wang in the global optimization framework to control the robotic swarm. SI techniques have broad application domains in science and engineering. Therefore, SI is considered a new branch of AI. Hence, modelling the collective behaviour of social swarms in nature like bird flocks, honey bees, ant colonies, and many more. Figure 1 shows an overview of the main properties of collective behaviour.. [5]

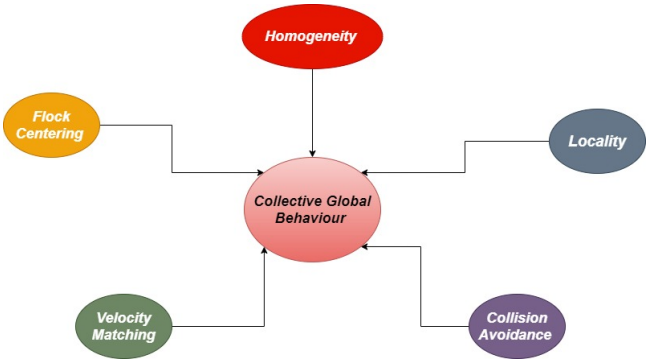

Fig. 1. The main Properties of modelled collective behaviour [5]

II. Dynamic Ant Algorithm

A. Decision Path

It is a record of the design decisions taken. Each path node represents a problem with the design. The starting point of the design process is the first knot of the decision path. A designer chooses all possible ways from the first node. The designer then comes up with a different design problem state (node). This selection process takes place multiple times until the designer is fully designed or is dead.

B. Decision graph

The structure is a choice for a design problem. Each node is a problem with the design. Every arc is a choice to get to the next node or state. It's identical to a decision tree, but the different selection sequence can come into the same state or node in this case. Each node may also have more than one degree. The initial state of the design problem is represented by the root node. The root node is known as the decision-making path to a leaf node.

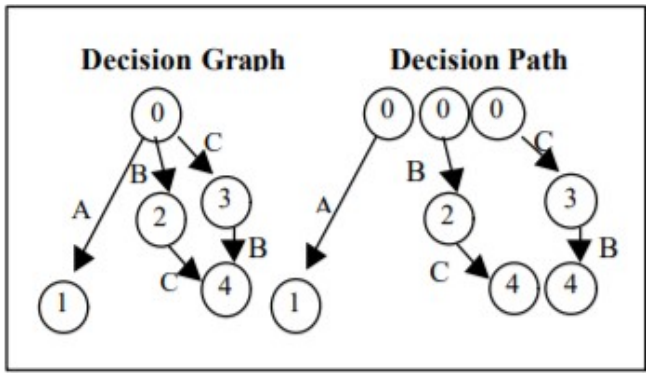

Fig. 2. Decision Tree for Graph and Path [6]

C. Construction of a decision path

This process makes a sequence of choices. The initial node, which is an first design state, starts this process. Then the options are listed, and the heuristic weight and the pheromone weight are mapped out in each possible choice as in the algorithm of Ant. An alternative is selected according to the state transition rule, which is explained in [6], and the algorithm then comes to the new state of the design problems. This process goes so far as to conclude the design process. The decision chart is also updated during the construction of the decision path.

D. State Transition Rule

It is a rule applied by selecting a choice from the weighted possible list to progress from one design step to the next. If you have selected all of the alternatives (by previous ants), the pheromone level biases will randomly choose the option. This is deduced from a different ACO. With ACO, the selection procedure is based on a linear combination of heuristic weights and the level of phéromones. However, in the case of Dynamic Ant, the heuristic weights are used until all choices are selected. Our method has the advantage that heuristic weight determinations are independent of the levels of pheromone.

E. Path Exploration

It's a process to start new paths from an exciting route. The process begins with the random scanning point selected on the track. An approach is copied as an initial path for a new ant from the root to the end of the investigation. The steps of the Dynamic Ant Algorithm now are presented. We call that Dynamic Ant as specific techniques are borrowed as explained in [6] This is a Dynamic Niche Sharing.

III. High Level Synthesis in Relation to Ant Colony Optimization(ACO

A. High Level Synthesis Flow

HLS Flow defines the process flow between conception and synthesis level. The design is carried out here in the form of an algorithm on an abstract class. In addition, the Register Transfer Level (RTL) is a tool used in generating a circuit, as described in [7] however, it can be simulated or verified if the circuit is then in the Hardware description language.

B. High Level Synthesis Formulation

Let v and w be arithmetic or logical operations and (v, w) be a precedence relation. This means that the output of v must be completed before w can start. Then the problem is choosing hardware h_v, h_w to execute$v, w$and scheduling each into a clock step s so that: $(t_w > t_v + l_V + M)\forall(w, v) \in P$ while minimizing a cost function $C(a, p, d)$ Here t_x is the initiation time of operation $xandl_vh$ is the latency (propagation delay) of operation v, executed on hardware unit h. P is the set of data precedence, and M is a timing margin that accounts for interconnect, mux and register delays. In the cost function, a refers to the total area, p to the power consumption and d to the total latency of the algorithm.

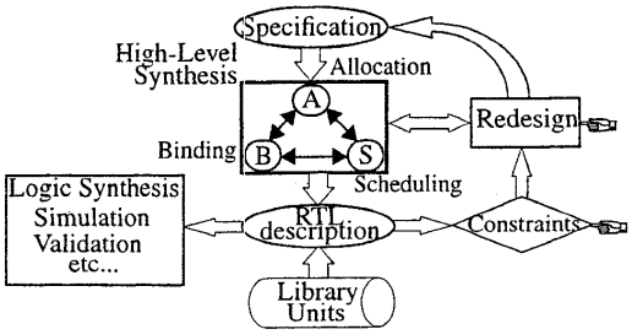

Fig. 3. HLS Flow [7]

C. High Level Synthesis Methods

1) High Level synthesis Steps: Three related problems are involved in a high-level synthesis: allocation, programming and binding [8]. Attribution means the assignment of sufficient hardware units for all operations. Scheduling is the time allocation for each operation in the detailed behavioural description without any violation of data precedence. Binding means assigning operations at certain times to hardware units. This is the stage in which all architectural interconnections are defined. These tasks are carried out simultaneously with the GA Synthesis (which represents the Ant Colony Optimization). Systems performing these functions in a row may require iteration between steps.

2) High Level Synthesis Constraints: In high-level synthesis, the main constraints of optimization are area and time. Speed (peak and average), throughput (and latency) and clock speed may also include testability, the ability to be routed (usually mux and bus area). The choice of technology, scaling, and design improvements vary in some respects. These constraints To accommodate such changes, a tool of synthesis should be flexible enough.

3) High Level Synthesis Scheduling Methods: The methods of scheduling in the HLS are below. Planning of ASAP/ALAP. As soon as all the necessary information is accessible, ASAP

referred to a schedule that assigns operations to be carried out. ALAP refers to the assignment of operations to be performed before their production values are necessary. Many methods are starting from these schedules. Heuristic scheduling, for example, list scheduling [9], orders operations based on priorities and force scheduling [10] that allocate operations based on distributed graphs derived from mobility operations—usually fast but not very flexible heuristic schedulers.Neural Networks scheduling has been implemented using self-organizing algorithms. The disadvantage of this approach is its speed. Simulated annealing scheduling generates random modifications in the schedule and accepts or rejects them according to an arbitrary rule and a parameter that gradually decreases with time. The disadvantage is the speed. Integer Linear Programming scheduling (ILP) is an exact scheduling technique. ILP scheduling is formulated as an optimization problem using certain constraints [7]. This method suffers from increasing time-complexity since it is exponential in the worst case. This puts more significant, more realistic problems out of their scope.

D. Genetic Algorithm Setup

The use of GA parameters is Type z of population: The complete circuit is a chromosome. There are 100,000 to several thousand typical population size for synthesis problems. The number of generations g maximum: This is the typical problem of tens of thousands of generations. In many cases, however, within one hundred, a good industrial solution is found. Typically, the crossover probability is between 20 and 60 per cent. Unless otherwise noted, the solutions shown in [7] use a 50 per cent overlap probability. Typically, the probability of mutation is between 0.1 and 1 per cent. It was set to 0,5per cent, for example, in this paper. The first circuits were generated randomly. For most examples shown in [7], a single point crossover was used. The method of selection was the roulette wheel that was replaced with a steady state. The halt criteria were the number of generations.

E. Chromosome encoding

The chromosome comprises two fields per operation (hw, cc), where hw is the field of the id of the FU used, and cc is the relative cycle for the start of the mobility-based operation. Mobility is from the start of the operation (ASAP) to the latest time interval, where an operation can be executed (ALAP). It is no longer than the critical path that ALAP allows.For small increases in time, minimal hardware solutions can often be found. Therefore, as resource-constrained as possible, we define and use an ARAP program to schedule the minimum possible hardware. The achievement of such a schedule has the same complexity as programming. Furthermore, this schedule can be relaxed to encode, and the upper limits can be used rather than the optimum tasks so that all reasonable solutions are represented in the design space.The number of bits required to represent a CDFG is given by: $n = \sum_{v=1}^{v} log_2 K_v + mob_v$. Where K is the number of units available that can perform mobility v. mob and V, the number of operations is total. The order of the operations in the chromosome is chosen to precede all of the operations in the graph. The objective function calculation thus reduces the complexity.

IV. HACO-F: An Accelerating High Level Synthesis Based Floating Ant Colony Optimization on FPGA

A. HACO-F Design based on High Level Synthesis

HACO-F is designed on HLS by integrating the Ant System (AS) algorithm, as the computation scale is more prominent than those of other ACOs. The HACO-F design is composed of two aspects. One is data optimization design, which can redefine the precision of variables for further reduction of resource usage. The other is loop optimization design, which makes the AS algorithm paralleled and integrated into an FPGA. The notations of HACO-F are explained below [2]

Notations	Specification
d	city location data
n	city quantity
m	ant quantity
NC	trip quantity
IN	pheromone initial value
seed	random function seed
ρ	volatilization coefficient
Q	constant coefficient for the calculating of $\Delta\tau$
η	heuristic information , the visibility of the road
tabu	visited cities list of ants
allowed	unvisited cites list of ants
L	tour length of all the ants
τ	road pheromone
$\Delta\tau$	update pheromone
p	transition probability
besttour	best tour map
tourlength	best tour length
α	parameter of τ
β	parameter of η
i	index variable of n
j	index variable of n
s	index variable of n
k	index variable of m

Fig. 4. Notations of HACO-F [2]

B. HACO-F Data Optimization Design

Data optimization is to redefine the precision of the variables used in the algorithm to reduce resource usage further. The process of data optimization mainly focuses on the boundaries of input variables, which may differ in different applications. According to the boundaries of input variables, it is easy to redefine the valuable bits of these input variables. Especially for float type variables, we redefine the precision by the following transition principle. For the precision insensitive variables, 6bits in decimal (20bits in binary) is defined, and the total bits should be defined smaller than 32bits (the same with float type). Otherwise, there would be little optimization or even worse. For some precision sensitive variables, the total bits may be up to 64bits (the same with double type). In particular, we may approximate the percentage values with

6bits in binary. Other variables, valuable bits, are defined based on the relations, such as the operations relations and valid bits dependency relations.

C. HACO-F Loop Optimization

Here in the Loop Optimization, we look at in brief how we can complete loops and functions with structured computational resources, and all this is done in parallel. In an FPGA, we have some Logic and available memory resources which can be used to facilitate a loop specifying an Optimization method. In this paper, just the main points will be sorted out. Nevertheless, [2] explains in more details how this method is applied to Optimize Loops and Functions effectively. So we are going to discuss briefly the method applied by [2]to optimize loops and functions. It is done in two stages: UNROLL for initializing and searching for Best solutions and PIPELINE to optimize all other loops.

a) step 1: Initialization process: Here all HACO-F are variables are initialized such as the input parameters, index variables. Also we have two variables which are a reciprocal to each other η and d, so at this point we have to design two loops, to avoid an overflow.And at the end we optimize the both loops

```
Set NC := 0; i := 0; j := 0; k := 0;
Check the input values scope;
Set n, m, ρ, Q , NCmax from input;
Initialize the random function with seed from input;
For i from 0 to n {
 #pragma HLS PIPELINE
  For j from 0 to n
   #pragma HLS PIPELINE
      Set d_ij from input;
}
For i from 0 to n {
 #pragma HLS PIPELINE
  For j from 0 to n {
   #pragma HLS PIPELINE
      Set η from d_ij;
      Set τ_ij := IN from input;
      Set Δτ_ij := 0;
    }
}
```

(a)

```
For NC from 0 to NC_MAX {
#pragma HLS UNROLL
   Search available solutions;
   Compute and compare the solutions;
   Update pheromones;
}
```

(b)

Fig. 5. (a)initialization of HACO-F (b) the best solution search [2]

The input variable seed is used to initialize the randomly generated function. The random numbers used in HACO-F are pseudo-random numbers generated by the mixed congruence method. We use the input variable seed as the initial value of the random function and generate the pseudo-random numbers by the multiplication-addition operator. Firstly, multiply seed with a predefined value, and we get the result as seed1. Secondly, add another predefined value to seed1, and then we get the first random number rand1. After that, set the random value rand1 as the new seed value, and go on as before. As seed can be set with a different value from input, the random numbers generated will be different at different times. In this way, we can get random numbers continually.Furthermore [2] explains how the programming variables.

b) Step 2: Search the best solutions. In this step, there are three parts inside. We unroll the outer loop to make the inner three parts execute in parallel, shown as Figure 5 (b), and describe the inner parts as three separate steps.

c) Step 3: Search available solutions. In this step, the outer loop is to finish the solution searching of all ants. And its upper bound is the ant number m. The first inner loop is to clear the tabu list of ant k. The following inner loop is to finish ant k's solution searching of all the cities. We pipeline these three loops, as there are temporal order relations among them, shown in Figure 6 (a). For optimization, we do some tricks for the implementation of the roulette rule. We merge the code of computing the transition probability (with in Figure 6 (b)), and the follow if conditional to one if conditional statement. We generate the random probability p_R as a random integer, which is more suitable for the process computation than percentage values. Then we multiply 127 on both sides of the inequality in the if conditional. So the inequality is transferred to $p - pro$*127 ¿= $p_sum * p_R$', where pR' is got from pR 0x7F. In this way, the computation of the transition probability with the '/' operator is removed, and it does not change the rule as stated by. [2]

```
For k from 0 to m {
#pragma HLS PIPELINE

   For i from 0 to n {
   #pragma HLS PIPELINE
      Clear tabu^k lists;
   }

   Place ant k on a random starting city;
   Place the starting city of the k ant in tabu^k(s);

   For s from 1 to n {
   #pragma HLS PIPELINE
      Move ant k to an unvisited city ";
      Insert the city to tabu^k(s);
      Update allowed^k
   }
}
```

(a)

```
Step 1. Compute the sum of transition probabilities
For j from 1 to n {
#pragma HLS PIPELINE
   Compute the sum of p_tabu^k(s)j; //variable p_sum
}
Step 2. Generate the roulette probability
Generate a random probability p_R
Step 3. Do roulette
For j from 1 to n {
#pragma HLS PIPELINE
 If j is in allowed^k {
   Compute the sum of p_tabu^k(s)j^k; //variable p_pro
   Compute the transition probability p = p_pro/p_sum ^k;
      If p is no less than p_R {
         roulette success;
         Break;
      }
 }
}
```

(b)

Fig. 6. (a)available solution search (b) transition rule [2]

d) Step 4: Compute and compare the solutions. In this part, there is one outer loop with two inner loops. The outer loop is to finish road length computations in the current cycle of all ants. The loop's scope is from 0 to m. The first inner loop is to compute ant k's road length. The following inner loop is to update the shortest road and shortest length when a

shorter length is found. We pipeline these three loops, shown in Figure 7 (a).

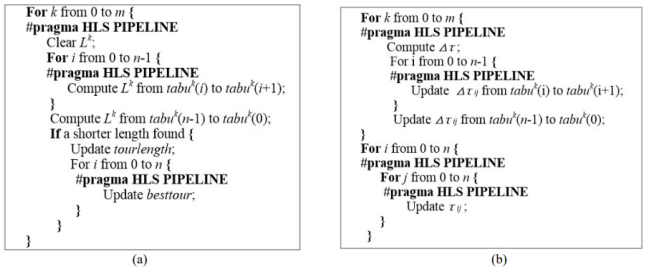

Fig. 7. (a)the comparison of solutions (b) road pheromone update [2]

e) Step 5: Update road pheromones based on new solutions. In this step, there are two loops. The first loop is to update each road's change in τ based on the new tour map of ant k. The update rule of change in $\tau$is based on the paper [11] with no simplification. The following inner loop is to update the road pheromone τ based on change in τ. As the precision of τ has been redefined, the values of τ can be confined automatically without the if-else conditional statement, which is a suitable method for code optimization. We pipeline these three loops, shown in Figure 7 (b).

f) Overall Architecture: The HACO-F IP core prototype implementation is based on the Zynq [12] FPGA platform, shown in Figure 8 (a). The PS part of Zynq is a dual-core ARM CPU, which can run a Linux OS for system control and data transfer control. The PL part of Zynq is the FPGA, where we can integrate the HACO-F IP core. The axilite [13] peripherals bus is applied to (a) (b) Figure 8.(a) the framework of the HACO-F prototype system (b) IP core of HACO-F integrate the HACO-F IP core and DMA as devices in the system, meanwhile to transfer variables' data. The DMA connects the HACO-F IP core and the PS high-performance bus ports through axis [13]protocol to directly access DDR memory. In this way, the HACO-F IP core is independent of the PS, leaving the PS resource to other processes. The IP core of HACO-F is shown in Figure 8 (b). Input variable d is a $B_n * B_n$ array, and we implement it as an axis slave port. The slave port is designed to connect to the axis master port of the DMA. Output variable best tour is also a B_n size array; we implement it as an axis master port and connect it to the axis slave port of the DMA. For all the other parameters described in data optimization, we use the axilite bus to finish the data transfer, as the data size is small. The bus width of both axis and axilite is 32 bits. In addition, the clock and reset ports are needed for the HACO-F IP core. After that, the interrupt port is used to indicate the end-of-run of HACO-F.

V. Ant Colony Optimization Algorithm For Fuzzy Controller And Design Of Its FPGA

A. Basic Concepts Of Fuzzy Controller and Ant Colony Optimization

a) Fuzzy Controller: The Ith Rule. Let i be an index for a fuzzy rule number, The ith rule denotes: R_i : If x_1(k) is

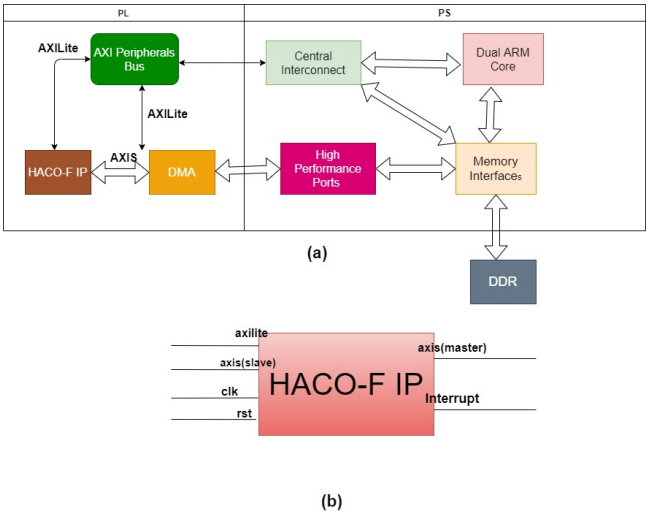

Fig. 8. (a)the framework of HACO-F prototype system (b) IP core of HACO-F [2]

A_{i1} And..... And x_n(k) is A_{in}. Then $u(k)$ is $a_i(k)$.

Also we have μ_A which is referred to as the membership degree for fuzzy set A. Assembling all these input together and considering that the fuzzy system consists of s fuzzy rules, the Weighted Average Deffuzzification method system output is computed by the equation below $u = \frac{\sum_{i=1}^{r} \phi_i(\vec{x}) a_i}{\sum_{i=1}^{r} \phi_i(\vec{x})}$. Here k is the time step,$x(k)$ is the input variable,$u(k)$ is the output action variable, A is a Fuzzy set,$a_i(k)$ is a recommended control action which is a fuzzy singleton. As analysed by [1], the inference engine, the fuzzy and Operation are implemented by the algebraic minimum of the fuzzy theory.$\vec{x} = (x_1, \ldots, x_n)$, is an input data set.There is also what we call the firing strength denoted by $\phi_i(\vec{x})$

b) Ant Colony Optimization(ACO): The ACO meta-heuristic was inspired by actual ant behaviour, particularly on how they forage for food. With these algorithms, a colony of artificial ants with a finite size is created, and each ant develops a solution. The performance measure is based on a quality function F(). ACO algorithms can be applied to problems described by a graph consisting of nodes and edges connecting the nodes. Optimization problem solutions can be expressed in terms of feasible paths on the graph. Among the feasible paths, ACO algorithms attempt to find the one with minimum cost. The problem of selecting fuzzy rule consequences can be described and solved by ACO algorithms. The data collected by ants are stored in the pheromone trails in association with edge links during the search process. The ants work together to find a solution by exchanging data through the pheromone trails. Edges also have the associated heuristic value , a preliminary piece of information on the definition of the problem instance or the run-times of information from a source other than ants. Ants can act simultaneously and independently, demonstrating cooperation.

B. Fuzzy Controller Design By Ant Colony Optimization

Here we look at how the ACO is directly applied to the Fuzzy controller system. First we have the predecessor part which is partitioned well ahead of time with relative ease, then we have input variable, and the sum of fuzzy set combinations of this input variables.This sum corresponds to the fuzzy rules in the product space. Also we have to select the best candidate action from a total of fuzzy controller combinations together with each r fuzzy rule. This shows us that there is a combinational optimization problem all this is properly analysed in [1]. This leads to a problem of combination optimisation. Because the ACO algorithm is proposed using the ant population to solve discrete problems in combination, this motivates the ACO proposal for the design of FC parts known as ACO-FC. ACO-novelty FC's is the proposed definition of a new pheromone matrix and heuristic values for FC design. This means that the research on an entirely new ACO algorithm in ACO-FC is not in focus. Trip determination from the current fuzzy rule to the following fuzzy rule is based on pheromone trails. Searching for the best one among all fuzzy rule combinations is pheromone level based. In the ACO-FC system, a colony of N_a ants is generated and placed in the nest. The terminal condition in this paper is the total number of search iterations performed by the ant colony. According to the pheromone level, the ant moves through a trip that corresponds to the choice of all fuzzy rule consequent actions. The pheromone level is stored in a matrix shown in Fig. 9. The special value τ_{0j} determines the control action selection of R1 when an ant moves out from its nest. It is known that control voltage increase causes a higher temperature. For the candidate control actions in each fuzzy rule, two heuristic values are assigned to two sets of control actions: positive and negative.As [1] would go further to explain, determination of the heuristic value usually requires a priori information about the problem instance. The controlled plant model is assumed unknown, except for the change of output direction with a control input. For example, in a temperature control system, it is known that the increase in control voltage causes the temperature to rise.

C. Hardware Implementation Of Ant Colony Optimization in ACO-FC

The ACO algorithm in the proposed ACO-FC is hardware implemented by FPGA. A PC simulates the designed FC and the controlled plant to test the design ACO chip performance. Configuration is considered online control. Like a practical plant online control, performance evaluation data are available only when the control starts. That is, at a time, only one Ant can be evaluated. Thus, the selection and evaluation functions of each Ant in the proposed ACO chip are implemented sequentially. The input from PC to FPGA is the signal F, denoting performance measure and representing eight bits. The FPGA output is the signal rule$[k]$ $[m]$, which sends information on the selected consequent action for fuzzy rule m by the kth Ant. The architecture of FPGA-implemented ACO is shown in Fig. 10. It consists of five modules. The summation, random number generator, and consequent selection modules select the FC and activate each new ant path-building process. The best ant selection module is activated for each new ant path-building process, but it works after controller evaluation in PC. Pheromone matrix update is performed in the pheromone matrix module and is activated only once in each iteration. That is, it is activated after all ants have built their paths. For illustration, assume that the candidate action number N is 16. The signal rule[k][m] is represented by four bits representing one of the 16 candidate actions. Detailed descriptions of the five modules are as follows.

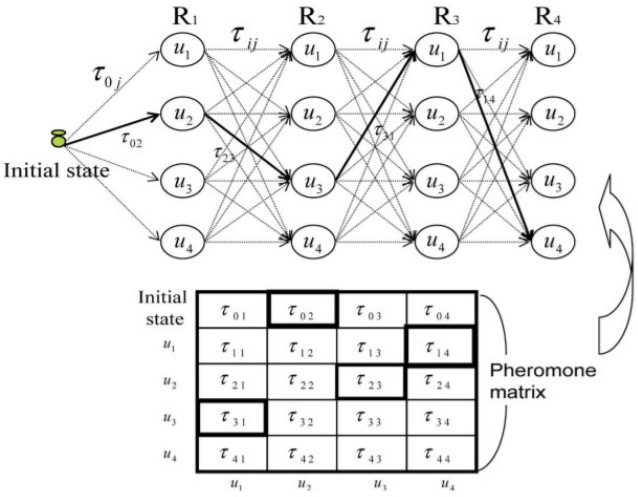

Fig. 9. FC constructed by an Ant trip and the corresponding pheromone matrix [1]

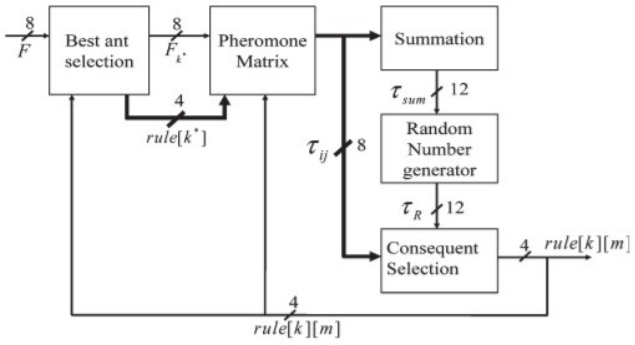

Fig. 10. Architecture of Hardware-Implemented ACO [1]

VI. Conclusion

In this paper, a Brief analysis has been done on Ant Colony Optimization, Looking at the general nature of the Algorithm. Furthermore, we have seen briefly how we can adapt the Ant Colony System to High-Level Synthesis. We discovered how effective the ACO-FC was performance-wise compared to other metaheuristic algorithms and is easier to implement an FPGA. The experiment and results of this study could

be seen in [1]. Furthermore, with the proposed HACO-F Algorithm, the loop optimization was explored properly by [2] in High-Level Synthesis, and results showed that the HACO-F Algorithm could accelerate more than six times the speed of an Ant System(AS)

REFERENCES

[1] C.-F. Juang, C.-M. Lu, C. Lo, and C.-Y. Wang, "Ant colony optimization algorithm for fuzzy controller design and its fpga implementation," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 3, pp. 1453–1462, 2008.

[2] S. Zhang, Z. Huang, W. Wang, R. Tian, and J. He, "Haco-f: An accelerating hls-based floating-point ant colony optimization algorithm on fpga." *International Journal of Performability Engineering*, vol. 13, no. 6, 2017.

[3] M. Dorigo and T. Stützle, *Ant Colony Optimization Theory*, 2004, pp. 121–152.

[4] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on evolutionary computation*, vol. 1, no. 1, pp. 53–66, 1997.

[5] A. Mohammadi and S. H. Zahiri, "Analysis of swarm intelligence and evolutionary computation techniques in iir digital filters design," in *2016 1st Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, 2016, pp. 64–69.

[6] R. Keinprasit and P. Chongstitvatana, "High-level synthesis by dynamic ant," *International journal of intelligent systems*, vol. 19, no. 1-2, pp. 25–38, 2004.

[7] E. Torbey and J. Knight, "High-level synthesis of digital circuits using genetic algorithms," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 1998, pp. 224–229.

[8] G. De Micheli, *Synthesis and optimization of digital circuits*. McGraw Hill, 1994, no. BOOK.

[9] R. A. Walker and R. Camposano, *A survey of high-level synthesis systems*. Springer Science & Business Media, 2012, vol. 135.

[10] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661–679, 1989.

[11] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[12] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[13] M. Ramirez, M. Daneshtalab, J. Plosila, and P. Liljeberg, "Noc-axi interface for fpga-based mpsoc platforms," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 479–480.