Ant Colony Optimization using High Level Synthesis

Bin Zaprunnizam   
Muhammad Amirul Hakimi   
Team ResQ Robot Hochschule Hamm Lippstadt Lippstadt, Germany muhammad-amirul-hakimi.bin-zaprunnizam@stud.hshl.de

*Abstract*— Ant Colony Optimization (ACO) has recently gained a lot of traction, and it is usually a highlight of major algorithms. The ACO is a probabilistic method that is used to find optimization routes. It was based on the behaviors that ants engage in when searching for food. Biological ants employ pheromone-based communication as their primary mode of communication. In this paper we will see how the ACO algorithm is a probabilistic approach for addressing computational problems that may be simplified to finding optimal pathways across graphs in computer science and operations research. Multi-agent approaches inspired by the behavior of actual ants are referred to as artificial ants. Combinations of artificial ants and local search algorithms have emerged as the preferred solution for a variety of optimization tasks. ACO is a type of simulated evolutionary algorithm that, according to prior study, provides several advantages.

# Introduction

Genetic Algorithm (GA) have been used to develop computer programs for specific objectives as well as build other computational structures. The research on Genetic Programming (GP) has sparked a recent resurgence in interest in AP with GA (GP). The GP paradigm enables program induction by searching a range of potential computer programs for an individual computer program that is well suited to solving or approaching the problem at hand. The genetic programming paradigm allows the evolution of computer programs that can perform alternative computations based on the results of intermediate calculations, that can perform computations on a variety of variables, that can perform repetitions and recursions to achieve the desired result, that can define and use computed values and subprograms, and that are not predefined in size, shape, or complexity. Because such mechanisms build hierarchical structures that would ease the generation of new high-level primitives from built-in low-level primitives, GP uses relatively low-level primitives that are defined independently rather than merged a priori into high-level primitives.

Unfortunately, because every real-life problem is a dynamic problem with complicated behaviors, GP has major flaws, one of which is random systems. Ant Colony Optimization (ACO) is the culmination of Dr. Marco Dorigo's research on computational intelligence techniques to combinatorial optimization, which he co-authored with Alberto Colorni and Vittorio Maniezzo [6]. ACO is based on an iterative process in which a population of simple agents construct candidate solutions repeatedly; this method is probability guided by heuristic information on the given problem instance as well as a shared memory containing experience gathered by the ants in previous iterations. Heuristic information about the present issue instance, as well as a shared memory including experience collected by the ants in past iterations, direct the process probabilistically. A component y is included in a state while a new solution is constructed, with a probability proportional to the attractiveness of the transition between the last component included in the state and y itself. The fundamental concept is to employ self-organizing principles to coordinate populations of artificial agents working together to solve issues. Self-organization is a collection of dynamical mechanisms through which structures emerge at the system's global level from interactions between its lower-level components. The rules governing interactions among the system's component units are implemented using just local data, with no reference to the global pattern, which is an emergent aspect of the system rather than one imposed by an outside ordering factor.

# Model

There are several iterations in the ACO algorithm. In each cycle, a group of ants uses heuristic knowledge and prior ant populations' experiences to develop comprehensive solutions. The pheromone trail, which is deposited on the constituent parts of a solution, is used to symbolize these gathered experiences. Depending on the problem to be solved, the pheromone can be placed on the components and/or connections in a solution. The following is a description of the pheromone update rule technique. In the ACO algorithm, an ant is a basic computational agent. It builds a solution to the problem one step at a time.

Chart, radar chart

Description automatically generated

Fig. 1. Map for Ant Colony Optimization

* Circles indicates ants
* Lines indicates paths have been taken by ants to find food

The diagram above shows how ACO works, a technique to calculate optimize path between point A nest and point B in space. In general point A is the nest and point B is the food. These are the steps how the model produced.

1. The first ant going to find food from point A to point B and comes back with leaving pheromone on the trail.
2. Second ant sense that pheromone knowing that it is the shortest path to find that food before the chemical evaporate. When the density of pheromone decreases it means that it is the longer path to find the food.
3. Most likely the next ants follow the line that has most intense pheromone sense.

The decision of choosing which paths or lines that are closest is depending on how dense the pheromone is. A permutation of π supplied objects can be used to explain any problem. The amount of pheromone released by ants in each round is calculated using a nxn pheromone matrix 𝜏ij. Any ACO algorithm begins by initializing the pheromone matrix, assigning an initial value to each pheromone entry (𝜏init > 0). At each cycle, Ants produces solutions π0... π1. Ants make several local judgments in order to construct solutions for succeeding item selections. The probability distribution over the choices of unchosen items in a certain selection set S is used to make decisions at random. S is determined in the following way.

Text

Description automatically generated

The relative weight of pheromone values and heuristic values is determined by the parameters α and β. The ACO algorithm performs several iterations until a given stopping condition is satisfied, such as a predetermined maximum number of iterations, a specific degree of solution quality, or the best solution has not changed over a specified number of iterations.

# High Level synthesis

Specification languages and design processes have evolved similarly in the hardware realm. Simulation tools have been widely used thanks to hardware description languages (HDLs) as Verilog (1986) and VHDL (1987). These HDLs have also been used as inputs to logic synthesis tools, allowing their synthesizable subsets to be defined.

An HLS tool performs the following task based on the high-level description of an application, an RTL component library, and certain design restrictions.

Diagram

Description automatically generated

Fig. 2. High Level Synthesis design steps

1. Compiles the requirements
2. Assigns hardware resources (functional units, processors, components for storage, buses, etc.)
3. Schedules activities according to clock cycles.
4. Associates activities with functional units.
5. Associates variables with data structures.
6. Ties bus transfers together.
7. Builds the RTL architecture

Sequential synthesis is based on the FSM model, whereas logic synthesis is based on the formalism of Boolean algebra. We enhance the FSM model by adding variable assignments for high-level synthesis. A collection of states, a set of transitions between states, and a set of actions connected with these states or transitions make up the FSM model. The behavioral description is compiled into an internal representation via the high-level synthesis mechanism. This model is used in all synthesis activities. There are 3 main elements in HLS which is allocation, scheduling, and binding.

* **Allocation** - The kind and amount of hardware resources (for example, functional units, storage, or connection components) required to meet the design requirements are defined by allocation. Some components may be introduced while scheduling and binding jobs, depending on the HLS tool. Connectivity components, such as buses or point-to-point connections between components, might be introduced before or after the binding and scheduling activities, for example. The RTL component library is used to pick the components. For each action in the specification model, it's critical to choose at least one component. Other synthesis activities will need to leverage the library's metrics and component attributes (such as area, delay, and power).
* **Scheduling** - All the operations in the specification model must be planned in cycles. To put it another way, for each operation like a = b op c, variables b and c must be read from their sources (either storage components or functional-unit components) and brought to the input of a functional unit capable of executing operation op, and the result a must be brought to its destinations (storage or functional units). The operation can be scheduled in a single clock cycle or over numerous clock cycles, depending on the functional component to which it is mapped. It's possible to link operations (the output of an operation directly feeds an input of another operation).
* **Binding** - A storage unit must be assigned to each variable that carries values across cycles. Multiple variables with nonoverlapping or mutually exclusive lives can also be bound to the same storage units. Every operation in the specification model must be assigned to one of the functional units that may perform it. If numerous units have this capability, the binding algorithm must choose the best one. Connectivity binding is also required for storage and functional unit binding, as each transfer from component to component must be bound to a connection unit such as a bus or multiplexer.

As I mentioned as paragraph above, the ACO algorithm can solve any problem requires to find shortest path with minimum time. Here in this paper, I will explain how ACO works in High Level Synthesis by using FPGA Routing with minimum CPU time. The use of Boolean-based routing to solve routing problems in FPGA layout is a relatively new concept. Any satisfactory assignment to the variables of the routing Boolean function indicates a legal routing solution in the Boolean based routing issue, which is satisfiable if the layout is routable else routing option is not considered. Recent developments in SAT solving algorithms and efficient implementation approaches have significantly increased capacity of solving routing task of FPGA’s and improved efficiency.

## FPGA Layout

In this experiment, the Xilinx 4000 is a conventional island type FPGA design. This design is made up of a two-dimensional array of customizable logic blocks (CLBs), connection blocks (CBs), and switching blocks (SBs). The combinational and sequential logic that performs the operation of a circuit is contained in each CLB labelled L. The routing resources are formed by programmable switches in the C and S blocks. C blocks use programmable switches to link CLB pins to channels. Signals can either travel through or through S blocks, which are encircled by C blocks. Three parameters W, Fc, and Fs are used to define the routing capacity of an FPGA design. The number of lines in a vertical or horizontal channel is determined by the channel width W. The adaptability of the C blocks the number of channels that each logic pin may connect to is specified as Fc. The S block's adaptability the number of additional paths that each wire segment enters a S block is denoted by Fs. Fs=3 because each wire section entering this S block can link to one track on each of the three other sides. Fc=2 because each logic pin can be linked to any two tracks in the C block.

Diagram, engineering drawing

Description automatically generated

Fig. 3. Island Style FPGA Model

The method of transforming the routing job into an atomic Boolean function that is satisfiable (has an assignment of input variables such that the produced function evaluates to constant "1") if and only if the design is routable is known as Boolean SAT-based routing. Any satisfying assignment to the binary variables of the Boolean function gives a legal routing solution. In this method Boolean routability function R (X) [25], has been formulated where X is a suitable Boolean vector of binary variables that encode the track number for each two-pin connection, can be expressed as the conjunction R(X ) = L(X ) ∧ E(X ) . Liveness constraint function L(X) guarantees that at least one global route alternative per two-pin connection should be chosen as a final legal routing solution. Exclusivity constraint function E(X) ensures that electrically distinct nets with overlapping vertical or horizontal spans in the same channel are always assigned to different track [5]. Boolean variables represent each routing alternatives of a netlist that represent all of the detailed routes [see Fig.3 (b)]. In this problem for NET A, there are only three possible detailed routes indicated by the three Boolean variables AR0, AR1, AR2 [26]. NET B and C are designed with their routes and the corresponding variables. A particular route is validated as the routing solution if its corresponding Boolean variable is assigned the logic value 1[see Fig.3 (b)]. For a netlist with n two-pin connections, Liveness constraints yield a set of n CNF clauses, each containing Fc positive literals. A single Boolean function represents all Liveness and exclusivity constraints which gives the routability of a particular netlist where X is a vector of Boolean variables that represent the possible detailed routes for each of the nets.

NET A route Boolean variable (AR0,AR1, AR2)

NET B route Boolean variable (BR0, BR1, BR2)

NET C route Boolean variable (CR0, CR1, CR2)

## Solve Using ACO Algorithm

To solve the FPGA routing issue using ANT colony optimization, the problem is represented as a directed graph, G = (V,E), with two vertices, I j V, that correspond to the two input options that the input may adopt, 0 or 1. The circuits' major inputs are then arranged, and vertices are linked to vertices representing nearby pins. With the starting weights set to 1, each edge e E is assigned a weight that indicates the quantity of pheromone that an ant may release on the routing path. Ants use a probabilistic strategy to explore the graph, with each calculation resulting in a potential routing vector.

The stages for our ACO framework are shown in the diagram below.

Diagram

Description automatically generated

Fig. 4. Graph showing initial weights assigmes to the routing paths

1. Create all of the SAT questions and set iterCount=2M (where M is the number of nets in an FPGA circuit). An MxM pheromone matrix encodes the pheromone information.
2. Repeat until iterCount does not equal zero.
3. Select a selection of SAT problems and create ants to symbolize each one.
4. Simulate ant movement and choose a subset of ants to place in the queue.
5. Create a routing design and simulate the logic. Permutations depending on the number of nets can be used to determine the routing pattern.
6. Check pheromone levels. By multiplying the pheromone values by an evaporation factor of 1, the pheromone matrix is updated.
7. Stop after all the SAT problems have been solved. Otherwise, decrease iterCount and go to the next step.

For the circuits given in Table 1, we examined the efficiency of ANT Colony methods. In our method, we assume that global circuit routing and placement are known. In our tests, the S block flexibility was modified from 3 to 3\*w, and the C block flexibility was set to Fc=W, allowing the CLB pin to link to any number of tracks. It displays ANT Colony routing findings for circuits that are large enough to be implemented on FPGAs. The benchmarks were obtained from. The experiments were done out using a C++ software running on a Sun Ultra SPARC-III processor in the Sun Blade workstation design. This project used the Xilinx 4000 series architectural model. The results of the experiment reveal that our method to FPGA routing utilizing ant colony optimization outperforms previous SAT-based FPGA routing solvers. Experiments indicate that the ANT Colony method can solve the FPGA routing issue with the least amount of CPU time and the fewest number of tracks per channel, outperforming alternative satisfiability-based detailed routers. Also routing conflicts among different routing solutions can be removed by representing routing solutions parallel using Ants parallelism techniques. For Alu2 Circuit, ACO has taken only 12.6 sec to route as compared to Grasp and Zchaff which has taken 26.52 sec and 24.38 sec to route respectively. Different alternatives for particular solution are represented parallel to solve the conflicts among different solutions

Chart, line chart

Description automatically generated

Fig. 9. Showing the comparison among ANT colony, GRASP, Zchaff for taking minimum CPU time to route a particular circuits.

# Conclusion

The ACO algorithm is used in this research to increase the performance of FPGA routing. Our findings suggest that the ACO algorithm is the best approach for routing FPGA chips because it requires the least amount of CPU time (in seconds). Our method works by having a group of agents work together to investigate various pathways. To effectively undertake this research, a stochastic decision-making technique is developed that combines global and local heuristics. ACO has routed the channel with the fewest possible tracks in all of the circuits used in our experiment. When compared to other traditional algorithms such as GRASP and Zchaff, ACO performed better and routed the circuit with minimum channel width and CPU time in all of the circuits investigated in our study. Our approach is more effective at identifying near-optimal solutions and scales better as the size of the problem increases.

# References

1. Dominey-Howes D. (2007) Geological and historical records of tsunami in Australia. Marine Geology 239: 99-123 doi:10.1016/j.margeo.2007.01.010
2. S. Schrödl and S. Edelkamp, Heuristic Search: Theory and Applications. Morgan Kaufmann Publishers, 2011
3. I. Ahmad, 40 Algorithms Every Programmer Should Know. [S.l.]: Packt Publishing, 2020.
4. E. Magid, T. Tsubouchi, E. Koyanagi and T. Yoshida, "Building a Search Tree for a Pilot System of a Rescue Search Robot in a Discretized Random Step Environment", *Journal of Robotics and Mechatronics*, vol. 23, no. 4, pp. 567-581, 2011. Available: 10.20965/jrm.2011.p0567.