Global Journal of Researches in Engineering
General Engineering
Volume 12 Issue 4 Version 1.0 Year 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 2249-4596 & Print ISSN: 0975-5861

Study of Mathematical Model and Ant Colony Optimization (ACO)

By Pawandeep Chahal
CMJ University

Abstract - In this paper we define those mathematical notions and terms that are useful about ACO and the relationships between ACO and other frameworks for optimization and control. This chapter defines and discusses the characteristics of: (i) the combinatorial optimization problems addressed by ACO, (ii) construction heuristics for combinatorial problems, (iii) the equivalence between solution construction and sequential decision process (iv) the graphical tools (state graph and construction graph) that can be used to represent and reason on the structure and dynamics of construction processes.

GJRE-J Classification : FOR Code : 230202

Strictly as per the compliance and regulations of :

Study of Mathematical Model and Ant Colony Optimization (ACO)

Pawandeep Chahal

Abstract - In this paper we define those mathematical notions and terms that are useful about ACO and the relationships between ACO and other frameworks for optimization and control. This chapter defines and discusses the characteristics of: (i) the combinatorial optimization problems addressed by ACO, (ii) construction heuristics for combinatorial problems, (iii) the equivalence between solution construction and sequential decision process (iv) the graphical tools (state graph and construction graph) that can be used to represent and reason on the structure and dynamics of construction processes.

I. Combinatorial Optimization Problems

ACO is a metaheuristic for the solution of problems of combinatorial optimization.

Instance of a combinatorial optimization problema : An instance of a combinatorial optimization problem is a pair (S, J), where S is a finite set of feasible solutions and J is a function that associates a real cost to each feasible solution, J: S → R. The problem consists in finding the element $s^* \in S$ which minimizes the function J:

$$s^* = \arg\min_{s \in S} J(s) \qquad (1)$$

Hereafter only sets S with finite cardinality will be considered, even if the above definition could be extended to countable sets of infinite cardinality. Given the finiteness of the set S, the minimum of J on S indeed exists. If such minimum is attained for more than one element of S, it is a matter of indifference which one is considered.

Combinatorial optimization problema : A combinatorial optimization problem is a set of instances of an optimization problem. The set of instances defining an optimization problem are usually all sharing some core properties or are all generated in a similar way. Therefore, an optimization problem defines a classification over sets of instances. This classification can be made according to several criteria that are usually based on both mathematical and practical considerations.

Static and dynamic optimization problems : Static combinatorial optimization problems are such that the value of the mapping J does not change during the execution of the solving algorithm. In dynamically changing problems the mapping J changes during the execution of the algorithm, that is, J depends on a time parameter t: J ≡ J (s, t).

If the statistical processes according to which the costs change over time are known in advance, then the optimization problem can be stated again as a static problem in which J is either a function of the time or has a value drawn according to some probability distribution. In these cases the minimization in Equation 1 has to be done according to the J's characteristics (e.g., minimization of the J's mean value, if J's values are drawn from a unimodal parametric distribution). On the other hand, when only incomplete information is available about the dynamics of cost changes, the problem has to be tackled online using an adaptive approach.

The set of problems here labeled as "static" are actually most of the problems usually considered in combinatorial optimization textbooks (e.g., the traveling salesman problem, the quadratic assignment problem, the graph coloring problem, etc.). They can be solved offline, adopting either a centralized or a parallel/distributed approach according to the available computing resources. Dynamic problems are somehow real-world versions of these problems. Routing in communication networks is a notable example of dynamic problem: the characteristics of both the input traffic and the topology of the network can change over time according to dynamics for which is usually hard to make robust prediction models. Moreover, in general routing requires a distributed problem solving approach.

In general terms, while for static problems using a centralized or a distributed algorithm is a matter of choice, dynamic problems usually impose more severe requirements, such that the nature (either centralized or distributed) of the problem has to be matched by the characteristics of the algorithm. In this sense, ACO's design, relying on the use of a set of autonomous agents, appears as rather effective, since it can be in principle used in both centralized and distributed contexts with little adaptation.

Primitive, environment, and constraints sets : An optimization problem can be formally identified in terms of a primitive set K, an environment set E, a solution set S, and a cost criterion J defined on S. The primitive set defines the basic elements of the problem. The environment set E is derived from the primitive set K as

Author : Research Scholar, CMJ University, Shillong, Meghalaya-793 003, Asst. Prof, Department of IT, Desh Bhagat Institute of Engg. & Management, Moga.

a subset of its power set, $E \subseteq P(K)$, and the solution set S is in turn derived from the environment set in terms of a family of subsets of E defined by a set of mathematical relations Ω among the K's elements, $S \subseteq P(E) \cap \Omega$. The set of relations Ω, which puts specific limitations on the way the elements in E can be selected in order to identify elements in S, is usually termed the constraints set.

The choices for sets K, E and Ω are not unique. Given an abstract definition of a problem, the same problem can be expressed in different ways according to different choices for these sets. One choice can be preferred over another just because it puts some more emphasis on aspects that are seen as more important in the considered context. In general, these facts raise the issue of the representation adopted to model the abstract problem under consideration in the perspective of attacking it with a specific class of algorithms. This issue is discussed more in depth in the following of this section. But before that, it is useful to make the above notions of primitive and environment sets more concrete through a few examples, and to introduce the notion of solution components which will play a central role throughout the thesis.

In order to explain in what a problem definition in terms of the sets K, E and Ω precisely consists of, let us consider the concrete case of two wide and quite general classes of combinatorial problems: matching problems and set problems [70], to which we will refer often throughout the thesis:

Matching problems : A matching in a graph is a set of edges, no two of which share a node. Goals in matching problems consist in finding either matching with maximal edge sets or, given that costs are associated to the edges/nodes, matching with minimal associated cost (weighted matching problems).

Matching problems in terms of primitive and environment sets : Let $K = \{1, 2, \ldots, N\}$ be a generic set of elements of interest, and let K be the primitive set. The environment and solution sets are derived as follows:

$$E = P(K)$$

$$S = \{\varepsilon \in E \mid \text{problem constraints } \Omega(K) \text{ are satisfied}\} \quad (2)$$

The expressions 3.2 mean that the solution set is directly defined in terms of subsets of K's power set. In the class of matching problems, of particular practical interest, as well as easier to solve, are those problems for which the underlying graph is a complete bipartite graph with two sets of nodes that are equal in size. Bipartite weighted matching of this type are also known as assignment problems, which are for instance the problems of assigning tasks to agents knowing the cost of making agent i deal with task j, and include important combinatorial problems like the TSP, the QAP and the VRP. Network flow problems can be also expressed in terms of generic bipartite matching. The following example shows in practice how K, E and S can be defined in the case of a TSP.

Example 1 :

Given an N cities TSP, $K = \{c_1, c_2, \ldots, c_N\} = \{1, 2, \ldots, N\}$ coincides with the set of the cities to be visited, $E=P(K)$ is the set of all their possible combinations, and S results from the application of Ω as the subset of elements in E which are cyclic permutations of size N. An alternative definition of K, E and Ω could consist in K being the set of pairs (p_i, c_j), $p_i, c_j \in \{1, 2, \ldots, N\}$, that is, the set of elements telling that city c_j is in position p_i in the solution sequence (notice that being the TSP's solutions cyclic permutations, the notion of position requires setting an arbitrary start city). In this case E is still the power set of K, but the syntax of the Ω relations is slightly different from before, S is in fact defined as

$$S = \varepsilon^k \in E, \varepsilon^k = \{(p_1^k, c_1^k), \ldots, (p_N^k, c_N^k)\} \mid \cup_i \; p_i^k = \{1, \ldots, N\} \wedge \cup_i \; c_i^k = \{1, \ldots, N\}$$

That is, the set of pairs must correspond to a permutation over {1, …, N}.

Set problems : In assignment problems solutions can be usually expressed in terms of ordered subsets of primitive elements, while in the case of set problems there is no explicit notion of ordering. Moreover, in most of the assignment problems the solution has a predefined size, while this is never the case for set problems. Set problems are also in general characterized by an additional level of complexity with respect to the assignment ones in the sense that is well expressed by the structure of the environment and solution sets:

Set problems in terms of primitive and environment sets : In set problems, which can be further classified in set covering, set packing and set partitioning problems, the corresponding of expression 3.2 takes the following form:

$$E = \{\varepsilon \in P(K) \mid \text{instance constraints } \Omega_I \text{ are satisfied}\}$$

$$S = \{\varepsilon' \in P(K) \mid \text{instance constraints } \Omega_I \text{ are satisfied}\} \quad (3)$$

These expressions point out the fact that the solution set is defined in a more complex way than in the matching case. Solutions are in this case sets of subsets of elements of the environment set, which, in turn, are subsets of elements of K. The Ω_I constraints that have been called a bit improperly "instance constraints" are defined by the actual characteristics of the instance at hand.

Instance of a combinatorial problem using a compact representation : Let C be a finite set of variables such that a solution in S can be expressed in terms of

subsets of C's elements. In particular, called X′ = P(C), S is identified by the subset of elements of X′ for which the relations in Ω are satisfied: S ⊆X′ ∩ Ω(C). Therefore, given the sets S, C and Ω(C), together with a real-valued cost function J (S), a problem of combinatorial optimization consists in finding the element s* such that:

$$s^* = arg \min_{s \in \{X' \cap \Omega(C)\}} J(s) \quad (5)$$

Following this representation, an instance of a combinatorial optimization problem can be also compactly represented by the triple

$$<C, \Omega, J> \quad (6)$$

The elements of C, which represent the object of the decisions of the optimization process, are called hereafter solution components.

a) Solution components

From above definition it is apparent that solution components always have a precise relationship with the primitive and environment sets. In particular, for assignment problems C coincides with K, while for set problems C coincides with E. However, here we prefer to speak in terms of solution components rather than primitive and environment sets, because of their more intuitive and general meaning of parts of which a solution is composed of:

Solution components : The solution components set C is operatively defined as the set from which a step-by-step decision process would select elements one-at-a-time and add them to a set x until a feasible solution is built, that is, until x ∈S.

According to this characterization, the notion of solution components plays a central role in this thesis, since combinatorial optimization is here framed in the domain of decision processes, and the components of a solution are precisely the step-by-step objects considered by the decisions processes. More specifically, ACO's target will consists in the learning of good decisions in the terms of pairs of components to be included in the building solutions.

Above definition implicitly implies that for each set C of solution components must exist a bijective mapping:

$$f_C : X \subseteq P(C) \rightarrow S, \quad (7)$$

such that each $s_i \in S$ has a finite subset $\{c_i^1, c_i^2, \ldots, c_i^n\} \in X$ of solution components as preimage in X, and this preimage is unique. That is, after a finite number of decision steps, where at each step t a new component c_t is included in the set x_t, the elements in $x_t \in X$ are expected to map through f_C onto an element $s \in S$. The characteristics of the mapping f_C define the level of correspondence between the problem under solution and the way solutions are represented. In particular, if f_C is not anymore surjective, not all the feasible solutions are going to have a preimage in terms of a single set of components. Such a choice could rule out the same possibility of addressing the optimal solution. On the other hand, if f_C is not anymore injective, the same solution in S can be addressed by one or more distinct elements in X. Such a choice would result in a sort of blurred image of the solution set as seen from the component set, since several solutions could be seen as the same solution, making potentially difficult for an algorithm to act optimally. In general, when the mapping f_C is not anymore bijective the representation will undergo some loss of necessary information. That is, additional information must be added to a subset x ∈ X of C's elements in order to map it onto a solution.

It is clear that once a mapping f_C has been defined, solution components can be seen in more general terms as decision variables. At each solution construction step a decision variable c_t representing any convenient value is assigned. The only strict requirement consists in the fact that sets of decision variables can be eventually mapped bijectively onto a feasible solution. Since in some sense it is natural to explicitly associate decision variables to parts of a solution, in the following we will preferably use the term "solution components" instead of "decision variables".

Even if this latter would likely make clearer the intrinsic meaning of ACO's pheromone variables, which are precisely associated to pairs (c_i, c_j) of decision variables: decision c_j is taken, conditionally to the fact that decision c_i has been already issued, according to a probability value which depends on the value of the pheromone variable $\tau_{c_i c_j}$ associated to the pair of decisions. The way ACO is discussed in this thesis in terms of sequential decision processes, as well as the recent work of Chang et al. [13], where ACO, departing from the usual application to "classical" combinatorial optimization problems, is applied to the solution of generic MDPs (therefore, dealing with stochastic transitions after the issuing of a decision), strongly confirm this interchangeable view of pheromone variables as pairs of decision variables or solution components.

II. Construction Methods

ACO's ant-like agents independently generate solutions according to an incremental construction process. Therefore, the notion of construction algorithm is at the core of ACO. A generic construction algorithm is defined here as follows:

Construction algorithm : Given an instance of the generic combinatorial optimization problem in the form equation 3.5, an algorithm is said a construction algorithm when, starting from an empty partial solution $x_0 = \phi$, a complete solution s ∈S is incrementally built by adding one-at-a-time a new component c ∈ C to the partial solution.

The generic iteration (also termed hereafter transition) of a construction process can be described as:

$$x_j = \{c_1, c_2, \ldots, c_j\} \rightarrow x_{j+1} = \{c_1, c_2, \ldots, c_j, c_{j+1}\},\ c_i \in C,\ \forall i \in \{1, 2, \ldots, |\ C\ |\}, \quad (3.10)$$

where $x_j \in X' = P(C)$ is a partial solution of cardinality (length) j, $j \leq |C| < \infty$.

The partial solutions, that is, the set of all the possible configurations of solution components that can be encountered during the steps of the construction algorithm, coincides with elements of the environment set X′. As it has been previously noticed, the majority of these elements are such that, in general, they are not subsets of some feasible solution set. That is, without a careful step-by-step checking, the construction process is likely to end up in a partial solution that cannot be further completed into a feasible solution.

The algorithmic skeleton of a generic construction strategy is reported in the pseudo-code of the Algorithm 3.1.

1. procedure Generic construction algorithm()
2. $t \leftarrow 0$;
3. $x_t \leftarrow \phi$;
4. while ($x_t \in / S \vee \neg$ stopping criterion)
5. $c_t \leftarrow$ select component($C \mid x_t$);
6. $x_{t+1} \leftarrow$ add component(x_t, c_t);
7. $t \leftarrow t + 1$;
8. end while return x_t;

Algorithm 1 : A general algorithmic skeleton for a construction algorithm.

It is the duty of the construction algorithm to guarantee that a sequence of feasible partial solutions, defined as it follows, is generated during the process:

Feasible partial solution : A partial solution $x_j \in X'$ is called feasible if it can be completed into a feasible solution $s \in S$, that is, if at least one feasible solution $s \in S$ exists, of which x_j is the initial sub tuple of length j in the case of sequences, or, of which x_j is a subset in the case of sets. The set of the feasible partial solutions is indicated with $X \subseteq X'$.

It is understood that a process generating a sequence of feasible partial solutions necessarily ends up into a feasible solution. The set X of all feasible sets x_j is finite since both the set S and the cardinality of the set associated to each feasible solution s_i are finite. Moreover, $S \subseteq X$, since all the solutions s_i is composed by a finite number of components, all belonging to C. Each feasible partial solution x_j has associated a set of possible feasible expansions:

Set of feasible expansions : For each feasible partial solution x_j, the set $C(x_j) \in C$ is the set of all the possible new components $c_j \in C$ that can be added to x_j giving in turn a new feasible (partial) solution x_{j+1}:

$$C(x_j) = \{c_j \mid \exists\, x_{j+1} : x_{j+1} \in X \wedge x_{j+1} = x_j \oplus c_j\} \quad (10)$$

Where the operator $\oplus$ represents the strategy adopted by the construction algorithm to include a new component into the building solution. In general, the characteristics of the sets C strongly depend on the precise form of the operator $\oplus$.

The very possibility of speaking in terms of feasible partial solutions and feasible expansion sets is related to the possibility of checking step-by-step the feasibility of the partial solution in order to take a sequence of decisions that can finally take to a feasible solution. For reasons that will be more clear in the following, we make a distinction between the components of the algorithm managing the aspects of feasibility from those specifically addressed at optimize the quality of the solution(s) that will be built. In order to check step-by-step the feasibility of the building solution, we assume that a logical device can be made available to the construction agent:

Feasibility-checking device : By feasibility-checking device we intend any algorithm which, on the basis of the knowledge of the set S and/or of the constraint set Ω, is able to provide in polynomial time an answer concerning the feasibility of a complete solution and the potential feasibility of a partial solution.

From a theoretical point of view it is always possible to find such a polynomial algorithm in the case of NP-hard problems and in all the subclasses of the NP-hard one. However, even in the case of NP-hardness, which is the most common and interesting case, to allow a practical use of the device the polynomial order should be small. Generally speaking, the computations associated to the device should be light. When this is not the case, it can result more convenient to incur the risk of building a solution which is not feasible, that can be either repaired or discarded. For the class of problems considered in this thesis it is often possible to have at hand a computationally-light feasibility-checking device. In fact, it is usually easy to check step-by-step the feasibility of a constructing solution for assignment problems like the TSP or the QAP. However, for some scheduling or covering problems, this same task can result both more difficult and computationally expensive to accomplish. Moreover, in the case of max constraint satisfaction problems this is precisely the problem. However, the point is that here we will not focus on the design of strategies for smart or optimized ways of dealing with feasibility issues. Surely this will be an important part of the specific implementations, but we assume that in some sense this is not the most important part of the story, which is, on the contrary, the optimization of the quality of the final solution output by the algorithm.

Figure 3.1 shows in a graphical way the generic step of a construction process, pointing out all the important aspects and their reciprocal relationships in very general terms. The feasibility checking device which defines the set C (x_t) of feasible expansions for the current partial solution x_t is indicate with the Ω box, to

stress the role of either the constraints set Ω and/or the explicit knowledge of the solution set to accomplish this sub-task. The specific strategy of selection and inclusion of the new component ct is indicated by the decision block π. The dashe d contour lines show the actual subsets of components defining respectively the partial solution x_t and the set of feasible expansions C (x_t). The chosen component ct belongs to this last. The diagram shows the case in which a feasible solution $x_s \in S$ is eventually constructed. The decision strategy π is generically assumed as making use of at least the information contained in the partial solution in addition to C (x_t). A similar diagram will be shown for the specific case of the ACO's ant agents, in order to show the peculiarities of the ACO's design with respect to this generic one.

This issue of the feasibility of the final solution has put in evidence the fact that during a construction process the single decisions cannot be seen as independent. On the contrary, they are tightly related, since all the decisions issued in the past will constrain those that can be issued in the future. On the other hand, feasibility is only one aspect of the entire problem of building a solution. The equally, if not more, important aspect concerns the quality of the solution. It is evident that the same considerations on the dependence among the decisions apply also when quality is considered. In general, to optimize the final quality, each specific decision should be taken in the light of all previous decisions, that is, according to the status of the current partial solution. This can be seen at the same time as a constraint and an advantage: building a solution in a sequential way allows to reason on each single choice on the basis of an incremental amount of information coming from the past and also possibly looking into the future through some form of look ahead.

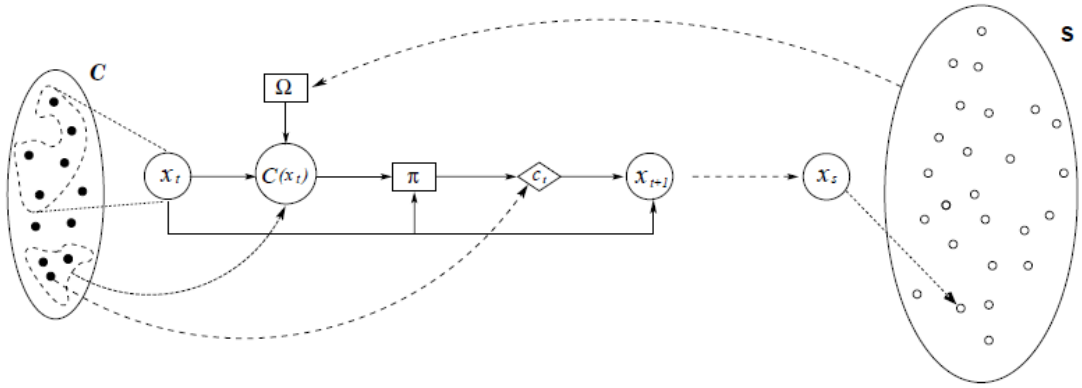

Figure 1 : The t-th step of generic construction process toward generation of a complete solution $x_s \in S$.

These are the basic key concepts to understand the rationale behind a large part of the con- tents of this chapter, which discusses construction and decision processes. In fact, in rather general terms, two construction strategies are going to be seen as different according to the different way of using and discarding the information contained in the partial solutions. In particular, it will be shown that an exact approach, like dynamic programming [3], makes use of the full information, while a heuristic approach, like ACO, drops off everything but the last included component.

III. Conclusion

In this paper we have defined the formal tools and the basic scientific background. That is, we have defined the terms and notions that will allow us to show important connections between ACO and other related frameworks and that will allow us to adopt a formal and insightful language to describe ACO.

More specifically the chapter has introduced the class of combinatorial optimization problems addressed by ACO and discussed the role and characteristics of different abstract representations of the same combinatorial optimization problem at hand. The chapter has also provided a formal definition and an analysis of construction methods for combinatorial optimization, made explicit and discussed the relationship between construction methods and sequential decision processes and, in turn, optimal control and defined the notion of construction graph as a graphical tool, derived from the state graph through the application of a generating function, which is useful to visualize and reason on sequential decision processes using a compact representation.

References Références Referencias

1. T. Abe, S. A. Levin, and M. Higashi, editors. Biodiversity: an ecological perspective. Springer-Verlag, New York, USA, 1997.
2. M. Beckmann, C.B. McGuire, and C.B. Winstein. Studies in the Economices of Transportation. Yale University Press, 1956.
3. R. Bellman. Dynamic Programming. Princeton University Press, 1957.
4. R. Bellman. On a routing problem. Quarterly of Applied Mathematics, 16(1): 87–90, 1958.

5. D. Bertsekas, J. N. Tsitsiklis, and Cynara Wu. Rollout algorithms for combinatorial optimization. Journal of Heuristics, 3(3):245–262, 1997.
6. M. Birattari, G. Di Caro, and M. Dorigo. Towards the formal foundation of ant programming. In M. Dorigo, G. Di Caro, and M. Sampels, editors, Ants Algorithms - Proceedings of ANTS 2002, Third International Workshop on Ant Algorithms, Brussels, Belgium, September 12–14, 2002, volume 2463 of Lecture Notes in Computer Science, pages 188–201. Springer-Verlag, 2002.
7. J.A. Bland. Optimal structural design by Ant Colony Optimization. Engineering optimization, 33: 425–443, 2001.
8. J.A. Bland. Layout of facilities using an Ant System approach. Engineering optimization, 32(1), 1999. [38] J.A. Bland. Space-planning by Ant Colony Optimization. International Journal of Computer Applications in Technology, 12, 1999.
9. K. Bolding, M. L. Fulgham, and L. Snyder. The case for chaotic adaptive routing. Technical Report CSE-94-02-04, Department of Computer Science, University of Washington, Seattle, 1994.
10. E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, 1999.
11. J. Broch, D.A. Maltz, D.B. Johnson, Y.C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom98), 1998.
12. D. Camara and A.F. Loureiro. Gps/ant-like routing in ad hoc networks. Telecommunication Systems, 18(1–3):85–100, 2001.
13. H.S. Chang, W. Gutjahr, J. Yang, and S. Park. An Ant System approach to Markov Decision Processes. Technical Report 2003-10, Department of Statistics and Decision Support Systems, University of Vienna, Vienna, Austria, September 2003.
14. C. Cheng, R. Riley, S.P. Kumar, and J.J. Garcia-Luna-Aceves. A loop-free extended bellman-ford routing protocol without bouncing effect. ACM Computer Communication Review (SIGCOMM '89), 18 (4):224–236, 1989.
15. L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In Proceedings of the Tenth National Conference on Artificial Intelligence, pages 183–188, 1992.
16. I. Cidon, R. Rom, and Y. Shavitt. Multi-path routing combined with resource reservation. In IEEE INFOCOM'97, pages 92–100, 1997.
17. I. Cidon, R. Rom, and Y. Shavitt. Analysis of multi-path routing. IEEE/ACM Transactions on Networking, 7(6):885–896, 1999.
18. A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL), 34:39–53, 1994.
19. M. Cottarelli and A. Gobbi. Estensioni dell'algoritmo formiche per il problema del commesso via ggiatore. Master's thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 1997.
20. J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the argentine ant. Journal of Insect Behavior, 3:159–168, 1990.
21. J.L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dynamics of collective sorting: robot-like ants and ant-like robots. In S. Wilson, editor, Proceedings of the First International Conference on Simulation of Adaptive Behaviors: From Animals to Animats, pages 356–365. MIT Press, Cambridge, MA, USA, 1991.
22. G. Di Caro and M. Dorigo. AntNet: A mobile agents approach to adaptive routing. Technical Report 97–12, IRIDIA, University of Brussels, Belgium, June 1997.
23. G. Di Caro, F. Ducatelle, and L.M. Gambardella. AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In Proceedings of Parallel Problem Solving from Nature (PPSN) VIII, volume 3242 of Lecture Notes in Computer Science, pages 461–470. Springer-Verlag, 2004.
24. G. Di Caro, F. Ducatelle, and L.M. Gambardella. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. European Transactions on Telecommunications, 16(5), October 2005.
25. E. W. Dijkstra. A note on two problems in connection with graphs. Numerical Mathematics, 1:269–271, 1959.
26. M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. Artificial Life, 5(2):137–172, 1999.
27. J.J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. IEEE/ACM Transactions on Networking, February 1997.
28. F. Glover. Tabu search, part I. ORSA Journal on Computing, 1:190–206, 1989.
29. J. Moy. OSPF version 2. Request For Comments (RFC) 1583, Network Working Group, 1994.
30. J. Moy. OSPF Anatomy of an Internet Routing Protocol. Addison-Wesley, 1998.
31. G. Owen. Game Theory. Academic Press, third edition, 1995.
32. M.L. Puterman. Markov Decision Problems. John Wiley & Sons, 1994.
33. Rome, Kay, and Friedemann Mattern. The design space of wireless sensor networks. IEEE Wireless Communications, 11(6):54–61, December 2004.