Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol 3, No.8, 2012

Dynamic Optimization of Network Routing Problem through Ant Colony Optimization (ACO)

Khalid Mahmood [1*], Shahid Kamal [1, 2], Hamid Masood Khan [1]

1. ICIT, Gomal University, D.I.Khan, (KPK), Pakistan
2. FSKSM, UTM, Johor Bahru, Malaysia

* Email of the Corresponding author: khalid_icit@hotmail.com

Abstract:

Search Based Software Engineering (SBSE) is a new paradigm of Software engineering, which considers software engineering problems as search problems and emphasizes to find out optimal solution for the given set of available solutions using metaheuristic techniques like hill climbing simulated annealing, evolutionary programming and tabu search. On the other hand AI techniques like Swarm particle optimization and Ant colony optimization (ACO) are used to find out solutions for dynamic problems. SBSE is yet not used for dynamic problems. In this study ACO techniques are applied on SBSE problem by considering Network routing problem as case study, in which the nature of problem is dynamic.

Keywords: SBSE, ACO, Metaheuristic search techniques, dynamic optimization

1. Introduction:

Swarm intelligence is a relatively new approach to problem solving that takes inspiration from the social behaviors of insects and of other animals. In particular, ants have inspired a number of methods and techniques among which the most studied and the most successful is the general purpose optimization technique known as ant colony optimization. Ant colony optimization (ACO) takes inspiration from the foraging behavior of some ant species. These ants deposit pheromone on the ground in order to mark some favorable path that should be followed by other members of the colony. Ant colony optimization exploits a similar mechanism for solving optimization problems (Dorigo, 2006).

Software Engineering often considers problems that involve finding a suitable balance between competing and potentially conflicting goals. There is often a bewilderingly large set of choices and finding good solutions can be hard. For instance, the following is an illustrative list of Software Engineering questions. The term Search Based Software Engineering was coined by Harman and Jones in 2001. In Search Based Software Engineering, the term 'search' is used to refer to the metaheuristic search-based optimization techniques that are used. SBSE seeks to reformulate Software Engineering problems as search-based optimization problems (or 'search problems' for short). This is not to be confused with textual or hyper textual searching. Rather, for Search Based Software Engineering, a search problem is one in which optimal or near optimal solutions are sought in a search space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions (M. Harman 2009).

Search based optimization has been applied to a wide variety of Software Engineering application areas including requirements engineering, project planning, cost estimation, maintenance, reverse engineering, refactoring, program comprehension, service-oriented software engineering and quality assessment. This area has come to be known as Search Based Software Engineering (M. Harman 2007).

The goal of this article is to utilize Ant Colony optimization (ACO) technique on dynamic problems. Section 1 provides some background and history of Search Based Software Engineering (SBSE) and Ant Colony Optimization (ACO) techniques. Section 2 provides an overview of Metaheuristic search techniques, Section 3 provides the related work in the field of ACO and SBSE, Section 4 provides the proposed methodology and proposed workflow model, whereas the Section 5 elaborates the conclusion.

1.1 Reformulating Software Engineering as a search problem.

The principal intention of this section is to demonstrate how conceptually simple is the reformulation of software engineering to search based software engineering. It is hoped that the reader is convinced that, at least in principal, it will be possible to apply metaheuristic search to large body of software engineering problems, where natural representations, fitness functions and operators suggest themselves.

In order to reformulate software engineering as a search problem, it will be necessary to define:

- A representation of the problem which is amenable to symbolic manipulation,
- A fitness function (defined in terms of this representation) and

- A set of manipulation operators.

Example, in the case of testing, the problem becomes one of searching for test cases, which satisfy some test-adequacy criterion.

1.1.1 Representation

The representation of a candidate solution is critical to shaping the nature of the search problem. Representations, which are frequently used in existing application for problem parameters are floating point numbers and binary code. In the later case, grey codes are generally preferred to 'pure binary' numbers. (M. Harman 2005), since successor decimals (for instance 7 and 8) are not neighbors in a pure binary code (4 mutations are required to mutate the four differing digits, rather than one). Grey codes avoid this problem.

1.1.2 Fitness Function

The fitness function is the characterization of what is considered to be a good solution. In measurement theoretical terms (N. Gold 2006), the fitness function need merely impose an ordinal scale of measurement upon the individual solutions it is applied to. That is, it will generally be sufficient to know which of the two candidate solutions is better according to the properties to be measured. The fitness function imposes a landscape, the characteristics of which will both determine which search techniques are most applicable and will shed light on the nature of the problem and its candidate solutions in terms of their perceived fitness. Fitness landscapes should not be too flat, nor should they have sharp maxima that can easily be missed. Often the fitness function, which first occurs, requires tuning to avoid these problems and to help guide the search towards good solution. In some cases it may be hard to define a fitness function, because the artifact to be optimized may have aesthetic qualities which make the determination of an ordinal scale metric difficult. However, in such situation a search based approach may still be applicable.

1.1.3 Operators

Different search techniques use different operators. As a minimum requirement, it will be necessary to mutate an individual representation of a candidate solution to produce a representation of a different candidate solution. Clearly, this is a very minimal requirement. It will make it possible to apply hill climbing approaches and certain forms of evolutionary computation. If it is also possible to determine the set of 'near neighbors' of a candidate solution (in terms of its representation) then simulated annealing and tabu search can be applied. If instead (or in addition), it is possible to sensibly cross over two individuals then genetic algorithms will be applicable.

2. Optimization Techniques

This section provides an overview of optimization techniques, focusing on those that have been most widely applied on software engineering. Space constraints only permit an overview. For more detail, the reader is referred to the recent survey of search methodologies edited by Burke and Kendall (E. Burke 2005). The section starts with classical techniques, distinguishing these from metaheuristic search. Hitherto, classical techniques have been little used as optimization techniques for software engineering problems; authors have preferred to use more sophisticated metaheuristic search techniques. However, there has been some work using classical techniques. Bagnall et al. (A. Bagnall 2001) applied a branch and bound approach to a formulation of the next release problem, while Barreto et al. (A. Barreto 2002) apply it to project staffing constraint satisfaction problems. Cortellessa et al. (Cortellesa 2004) use classical optimization techniques to address decision making problems in component sourcing, optimizing properties such as quality and reliability. Del Grosso et al. (Del Grosso 2006) use a combination of classical and metaheuristic techniques to test for buffer overflow.

2.1 Classic Techniques

Linear programming (LP) is a mathematical optimization technique that is guaranteed to locate the global optimum solution. The inputs to a linear programming model are a set {x1,...,xn} of n real, non-negative values, called the decision variables. The goal is to maximize the value of some linear expression in these decision variables subject to a set of constraints, expressed as linear equations in the decision variables. That is

$$\text{Maximize} \sum_{i=1}^{n} c_i x_i$$

Where {c1,...,cn} is a set of problem specific coefficients, subject to a set of m constraints of the form

$$\begin{gathered}\sum_{i=1}^{n} a_{1i}x_i \le b_1\\ \vdots \\ \sum_{i=1}^{n} a_{mi}x_i \le b_m\end{gathered}$$

Where a_{ij} and b_i are problem determined constants. The constraints can also be expressed using ≥ and = in place of ≤ and the goal can be minimization rather than maximization.

This formulation is typically applied to problems such as resource and plant allocation. It requires a clear cut determination of a single objective to be optimized and a set of well understood constraints that can be captured as a set of linear equations. If a software engineering problem can be formulated in this way, then LP is a good choice because there exist efficient LP optimization algorithms and the solution is guaranteed to be globally optimal. If some of the decision variables are further constrained to take on only integer values, then the result is a further constrained model. Integer programming models can capture a wider set of possible problem domains, because of this additional constraint. For example, it now becomes possible to model situations in which a decision variable is constrained to be a Boolean choice variable; yielding a value of either 1 or 0. However, the additional constraints can lead to model formulations that are far harder to solve than their linear programming counterparts. One other classical technique for optimization deserves mention in this section: branch and bound. This is an approach that seeks to tame the exponential explosion that is inherent in most search problems, by a simple iterative process of branching from a current solution, while simultaneously maintaining a set of bounds that prune the possible search space as it expands through branching.

2.2 Metaheuristic Search

This section provides a brief overview of three metaheuristic search techniques that have been most widely applied to problems in software engineering: hill climbing, simulated annealing and genetic algorithms.

2.2.1 Hill Climbing

Hill climbing (M. Harman 2000) starts from a randomly chosen candidate solution. At each iteration, the elements of a set of 'near neighbors' to the current solution are considered. Just what constitutes a near neighbor is problem specific, but typically neighbors are a 'small mutation away' from the current solution. A move is made to a neighbor that improves fitness. There are two choices: In next ascent hill climbing, the move is made to the first neighbor found to have an improved fitness. In steepest ascent hill climbing, the entire neighborhood set is examined to find the neighbor that gives the greatest increase in fitness. If there is no fitter neighbor, then the search terminates and a (possibly local) maxima has been found. Figuratively speaking, a 'hill' in the search landscape close to the random starting point has been climbed. Clearly, the problem with the hill climbing approach is that the hill located by the algorithm may be a local maxima, and may be far poorer than a global maxima in the search space. For some landscapes, this is not a problem because repeatedly restarting the hill climb at a different locations may produce adequate results (this is known as multiple restart hill climbing). Despite the local maxima problem, hill climbing is a simple technique which is both easy to implement and surprisingly effective (Mitchell 2002).

2.2.2 Simulated Annealing

Simulated annealing (Metropolis 1953) can be thought of as a variation of hill climbing that avoids the local maxima problem by permitting moves to less fit individuals. Simulated annealing is a simulation of metallurgical annealing, in which a highly heated metal is allowed to reduce in temperature slowly, thereby increasing its strength. As the temperature decreases the atoms have less freedom of movement. How ever, the greater freedom in the earlier (hotter) stages of the process allow the atoms to 'explore' different energy states. A simulated annealing algorithm (Bouktif 2006) will move from some point x1 to a worse point x0 with a probability that is a function of the drop in fitness and a 'temperature' parameter that (loosely speaking) models the temperature of the metal in metallurgical annealing. The effect of 'cooling' on the simulation of annealing is that the probability of following an unfavorable move is reduced. At the end of the simulated annealing algorithm, the effect is that of pure hill climbing. However, the earlier 'warmer' stages allow productive exploration of the search space, with the hope that the higher temperature allows the search to escape local maxima. The approach has found application in several problems in search based software engineering (Mitchell 2002).

2.2.3 Genetic Algorithms

Genetic algorithms (J. H. Holland 1975) use concepts of population and of recombination. Of all optimization algorithms, genetic algorithms have been the most widely applied search technique in SBSE, though this has largely been for historical reasons, rather than as a result of any strong theoretical indications that these approaches are in some way superior. A generic genetic algorithm (Clark 2003) is presented in Figure 1.

An iterative process is executed, initialized by a randomly chosen population. The iterations are called generations and the members of the population are called chromosomes, because of their analogs in natural evolution. The process terminates when a population satisfies some pre-determined condition (or a certain number of generations have been exceeded). On each generation, some members of the population are recombined, crossing over elements of their chromosomes. A fraction of the offspring of this union are mutated and, from the offspring and the original population a selection process is used to determine the new population. Crucially, recombination and selection are guided by the fitness function; fitter chromosomes having a greater chance to be selected and recombined. There are many variations on this overall process, but the crucial ingredients are the way in which the fitness guides the search, the recombinatory and the population based nature of the process. There is an alternative form of evolutionary computation, known as evolution strategies (Schwefel 1998), developed independently of work on Genetic Algorithms. However, evolution strategies have not been applied often in work on SBSE. An exception is the work of Alba and Chicano (Alba 2009), which shows that evolution strategies may outperform genetic algorithms for some test data generation problems.

```
Set generation number, m: = 0
Choose the initial population of candidate solutions, P (0)
Evaluate the fitness for each individual of P (0), F (Pi (0))
Loop
Recombine: P (m):= R (P (m))
Mutate: P (m):= M (P (m))
Evaluate: F (P (m))
Select: P (m+1):= S (P (m))
```

Figure 2.1: A Generic Genetic Algorithm

There is also a variation of genetic algorithms, called genetic programming (Koza 1992), in which the chromosome is not a list, but a tree. The tree is the abstract syntax tree of a simple program that is evolved using a similar genetic model to that employed by a genetic algorithm. Genetic programs are typically imperfect programs that are, nonetheless, sufficiently good for purpose. Fitness is usually measured using a testing-based approach that seeks to find a program best adapted to its specification (expressed as a set of in-put/output pairs). Genetic programming has been used in SBSE to form formulate that capture predictive models of software projects and in testing.

3. Literature Survey

The term Search Based Software Engineering was coined by Harman and Jones in 2001 (Harman 2001). SBSE tries to solve the problems associated with the balancing of competing constraints, trade-offs between concern and requirements imprecision.

SBSE have been applied to a number of software engineering activities right from requirements engineering (Bagnall 2001), project planning and cost estimation (Antoniol 2004), testing (Briand 2005), to automated maintenance (Bouktif 2006), service oriented software engineering (Canfora 20005), compiler optimization (Cohen 2006) and quality assessment (Bouktif 2006).

The Idea of Swarm Particle Intelligence was first proposed the French entomologist Pierre-Paul Grass´ e (Grass 1946). He observed that some species of termites react to what he called “significant stimuli”. He used the term stigmergy (Grass 1959) to describe this particular type of communication in which the “workers are stimulated by the performance they have achieved”.

Christian Blum (Christian 2005) in his paper explains the Basics of Swarm Intelligence and Ant Colonies social behavior, types of ants and their structure. The author elaborates the ACO algorithm with detail examples and its theoretical studies.

Marco Dorigo (Dorigo 2006) in his paper briefly describes the basic theme of Swarm Intelligence, its biological inspiration and Ant Colony Optimization Algorithm. The author applied the ACO on Traveling sales person (TSP) problem, metaheuristic optimization techniques including Simulated Annealing, Tabu search and on evolutionary algorithms. Furthermore the author justifies the ACO algorithms through some theoretical results. In this paper the author highlights the applications of ACO algorithm through NP-Hard problems, Telecommunication Networks (firstly on telephone network (Schoonderwoerd 1996), packet switched networks (Dicaro 1998) and further on Mobile networks (Ducatelle 2005), (DiCaro 2005) and its industrial applications. In the end the author list the hot issues related to ACO and points out some latest research areas in the ACO like dynamic optimization, stochastic problem, Multi objective optimization, Parallel implementation and finally continuous optimization issues.

Marco Dorigo (Dorigo 2000) in this technical report paper describes the history of Optimization algorithms from local search problems (greedy algorithms) to Ant colony optimization algorithm. In this report the author elaborate the ACO Algorithm, its application on static as well as dynamic problems. Author highlights the examples of ACO which includes, Single Machine total weighted tardiness scheduling problem, the generalized assignment problem, the set covering problem and Ant net for network routing problem. In the end the author highlights the future direction in the ACO algorithms by emphasizing on the Dynamic Optimization as the core area of research.

Zhi-hui Zhan (Zhan 2009) in his paper extends the study of ACO algorithm and proposes a new algorithm namely Discrete Particle Swarm Optimization (DPSO) for Multiple Destination Routing Problems, The author describes the algorithm and shows its empirical results comparisons with other comparative algorithms.

Guohui Zhang (Zhang 2009) in his paper applies Flexible job-shop scheduling problem (FJSP) as an extension of the classical job-shop scheduling problem. Although the traditional optimization algorithms could obtain preferable results in solving the mono objective FJSP. However, they are very difficult to solve multi-objective FJSP very well. In this paper, a particle swarm optimization (PSO) algorithm and a tabu search (TS) algorithm are combined to solve the multi-objective FJSP with several conflicting and incommensurable objectives. PSO which integrates local search and global search scheme possesses high search efficiency. The computational results have proved that the proposed hybrid algorithm is an efficient and effective approach to solve the multi-objective FJSP, especially for the problems on a large scale.

Zhi-Hui Zhan (Zhan 2009), in his paper, proposes an extension to Particle Swarm Optimization (PSO) algorithm through Adaptive Particle Swarm Optimization (APSO) Algorithm. This progress in PSO has been made possible by ESE, which utilizes the information of population distribution and relative particle fitness, sharing a similar spirit to the internal modeling in evolution strategies. Based on such information, an evolutionary factor is defined and computed with a fuzzy classification method, which facilitates an effective and efficient ESE approach and, hence, an adaptive algorithm. The author justifies his proposed work through statistical as well as empirical validation.

Yu Bin (Bin 2009) proposes an improved ant colony optimization (IACO), to vehicle routing problem (VRP), a well-known combinatorial optimization problem, holds a central place in logistics management. In this paper the author argues that IACO possesses a new strategy to update the increased pheromone, called ant-weight strategy, and a mutation operation, to solve VRP. The computational results for fourteen benchmark problems are reported and compared to those of other metaheuristic approaches.

4. Proposed Methodology:

All applications of SBSE concerned should be termed as ‘static’ or ‘offline’ optimization problems. That is, problems where the algorithm is executed off line in order to find a solution to the problem in hand. This is to be contrasted with ‘dynamic’ or ‘on line’ SBSE, in which the solutions are repeatedly generated in real time and applied during the lifetime of the execution of the system to which the solution applies. The static nature of the search problems has tended to delimit the choice of algorithms and the methodology within which the use of search is applied. Particle Swarm Optimization and Ant Colony Optimization techniques have not been used in the SBSE literature. These techniques work well in situations where the problem is rapidly changing and the current best solution must be continually adapted. For example, the paradigm of application for Ant Colony Optimization is **“dynamic network routing”**, in which paths are to be found in a network, the topology of which is subject to continual change. The ants lay and respond to a pheromone trail that allows them quickly to adapt to network connection changes.

4.1 Workflow Model for Dynamic Optimization

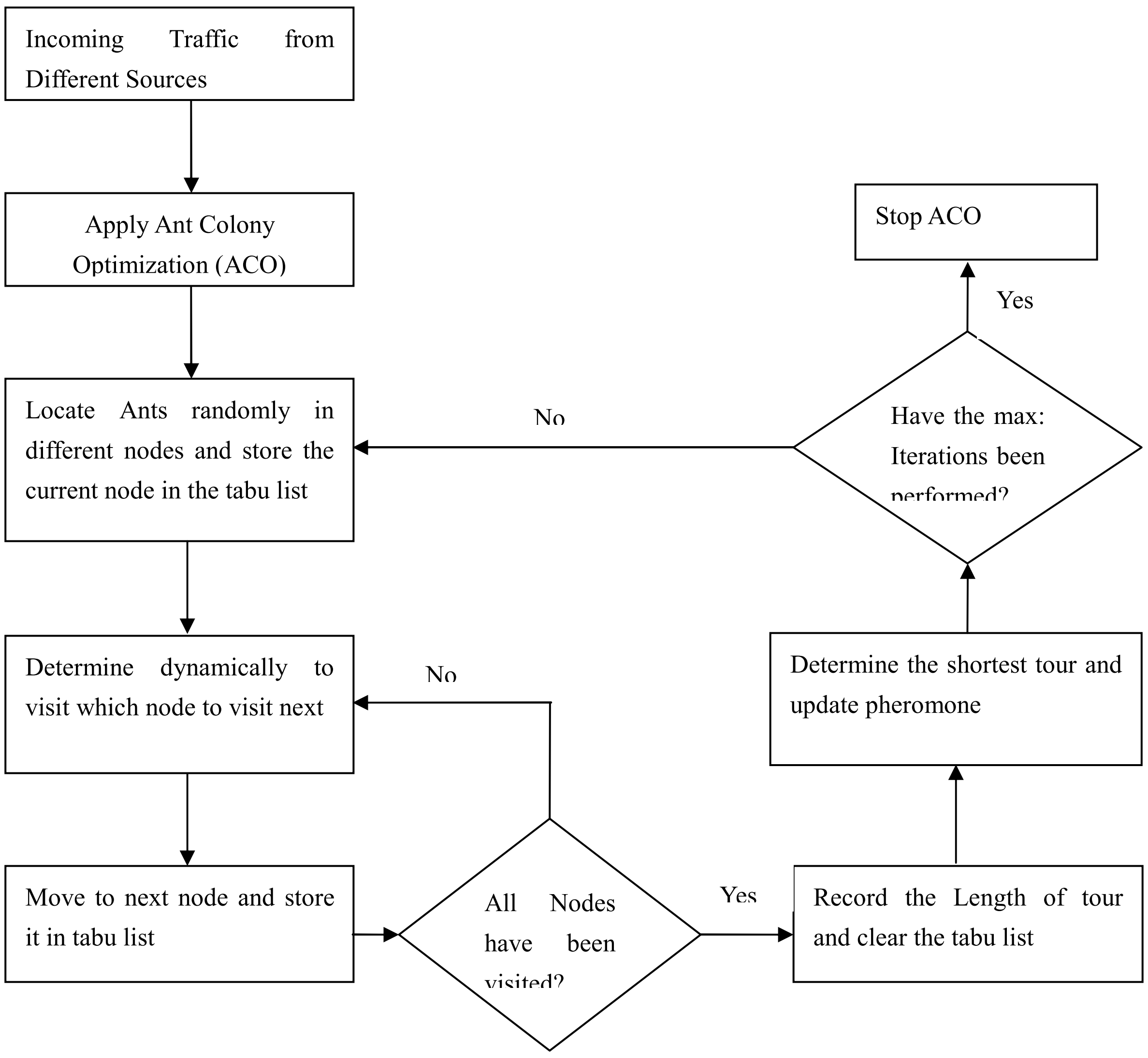
Incoming Traffic from Different Sources
Apply Ant Colony Optimization (ACO)
Locate Ants randomly in different nodes and store the current node in the tabu list
Determine dynamically to visit which node to visit next
Move to next node and store it in tabu list
All Nodes have been visited?
No
Yes
Record the Length of tour and clear the tabu list
Determine the shortest tour and update pheromone
Have the max: Iterations been performed?
No
Yes
Stop ACO

Figure 4.1: Workflow model for dynamic optimization

5. Conclusion:

Search Based Software Engineering (SBSE) is used for finding a near optimal solution for different software activities throughout software development life cycle. SBSE has been applied to problems having static nature, but yet not applied to problems having dynamic nature. On the other hand Swarm Particle Intelligence techniques, such as Ant Colony Optimization (ACO) uses ants behavior and structure to find the real world problems, using AI technique to SBSE dynamic search problem will probably find the objective which yet not been achieved. Dynamic Network Routing problem using ACO will yield this objective. Results will be statistically and empirically tested and compared with other competitive studies to validate the research being proposed.

References:

Marco Dorigo et al. (2006). Ant Colony Optimization, IEEE Computational Intelligence Magzine, November.

Mark Harman, S. Afshin Mansouri, (2009). Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications, Technical Report TR-09-03, April 9.

Mark Harman (2007). Automated Test Data Generation using Search Based Software Engineering, IEEE Computer Society, Second International Workshop on Automation of Software Test (AST'07).

M. Harman, S. Swift, and K. Mahdavi. (2005). An empirical study of the robustness of two module clustering fitness functions, In Genetic and Evolutionary Computation Conference (GECCO 2005), pages 1029–1036, Washington DC, USA, June, Association for Computer Machinery.

N. Gold, M. Harman, Z. Li, and K.Mahdavi. (2006). A search based approach to overlapping concept boundaries, In 22nd International Conference on Software Maintenance (ICSM 06), Philadelphia, Pennsylvania, USA, Sept. To appear.

E. Burke and G. Kendall. (2005). Search Methodologies, Introductory tutorials in optimization and decision support techniques, Springer.

A. Bagnall, V. Rayward-Smith, and I. Whittley, (2001). The next release problem, Information and Software Technology, 43(14):883–890, December.

A. Barreto, M. Barros, and C. Werner. (2001). Staffing a Software Project: A constraint satisfaction and optimization based approach, Computers and Operations Research (COR) focused issue on Search Based Software Engineering

V. Cortellessa, F. Marinelli, and P. Potena (2001). An optimization framework for "build–or–buy" decisions in software architecture, Computers and Operations Research (COR) focused issue on Search Based Software Engineering.

C. Del Grosso, G. Antoniol, E. Merlo, and P. Galinier. (2003). Detecting buffer overflow via automatic test input data generation, Computers and Operations Research (COR) focused issue on Search Based Software Engineering.

M. Harman, R. Hierons, and M. Proctor (2002). A new representation and crossover operator for search-based optimization of software modularization, In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pages 1351–1358, San Francisco, CA 94104, USA, 9-13 July. Morgan Kaufmann Publishers.

B. S. Mitchell and S. Mancoridis (2002). Using heuristic search techniques to extract design abstractions from source code, In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pages 1375–1382, San Francisco, CA 94104, USA, 9-13 July, Morgan Kaufmann Publishers.

N.Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller (1953). Equation of state calculations by fast computing machines, Journal of Chemical Physics, 21:1087–1092.

S. Bouktif, H. Sahraoui, and G. Antoniol (2006). Simulated annealing for improving software quality prediction, In GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, volume 2, pages 1893–1900, Seattle, Washington, USA, 8-12 July, ACM Press.

J. H. Holland (1975), Adoption in Natural and Artificial Systems, MIT Press, Ann Arbor.

J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Sheppard (2003). Reformulating software engineering as a search problem, IEEE Proceedings — Software, 150(3):161–175.

H. Schwefel and T. B¨ ack (1998). Artificial evolution: How and why? In D. Quagliarella, J. P´ eriaux, C. Poloni, and G. Winter, editors, Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, pages 1–19, John Wiley and Sons.

E. Alba and J. F. Chicano. *"Observations in using parallel and sequential evolutionary algorithms for automatic software testing*", Computers and Operations Research (COR) focused issue on Search Based Software Engineeering. To appear.

J. R. Koza (1992), *"Genetic Programming: On the Programming of Computers by Means of Natural Selection"* MIT Press, Cambridge, MA.

Mark Harman, Bryan F. Jones (2001), *"Search Based Software Engineering",* Information and Software Technology, 43 (2001), 833-839.

A. Bagnall, V. Rayward-Smith, and I. Whittley (2001), *"The next release problem"*. Information and Software Technology, 43(14):883–890, December.

G. Antoniol, M. Di Penta, and M. Harman (2004), *"A robust search based approach to project management in the presence of abandonment, rework, error and uncertainty*", In 10th International Software Metrics Symposium (METRICS 2004), pages 172–183, Los Alamitos, California, USA, Sept. 2004. IEEE Computer Society Press.

L. C. Briand, Y. Labiche, and M. Shousha (2005), *“Stress testing real-time systems with genetic algorithms*”, In Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005, pages 1021–1028, ACM.

S. Bouktif, G. Antoniol, E. Merlo, and M. Neteler (2006), *“A novel approach to optimize clone refactoring activity*”, In GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, volume 2, pages 1885–1892, Seattle, Washington, USA, 8-12 July, ACM Press.

G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani (2005), *“An approach for qoS-aware service composition based on genetic algorithms*”, In H.-G. Beyer and U.-M. O’Reilly, editors, Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, pages 1069–1075, ACM.

M. Cohen, S. B. Kooi, and W. Srisaan (2006), *“Clustering the heap in multi-threaded applications for improved garbage collection*”, In GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, volume 2, pages 1901–1908, Seattle, Washington, USA, 8-12 July, ACM Press.

S. Bouktif, H. Sahraoui, and G. Antoniol (2006), *“Simulated annealing for improving software quality prediction*”, In GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, volume 2, pages 1893–1900, Seattle, Washington, USA, 8-12 July, ACM Press.

P.-P. Grass´ e (1946), *‘ Les Insectes Dans Leur Univers ’* , Paris, France: Ed. du Palais de la d´ ecouverte.

P.-P. Grass´ e (1959), “*La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes Natalensis et Cubitermes sp. La th´ eorie de la stigmergie* ‘: Essai d’interpr´ etation du comportement des termites constructeurs,” Insectes Sociaux, vol. 6, pp. 41–81.

Christian Blum (2005), *“Ant Colony Optimization”*, GECCO, June 26, Washington, US.

Marco Dorigo et al. (2006), “*Ant Colony Optimization “Artificial Ants as a computaional intelligence techniques”,* IEEE Computational Intelligence Magazine, November.

R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz (1996), *“Ant-based load balancing in telecommunications networks,*” Adaptive Behavior, vol. 5, no. 2, pp. 169–207.

G. Di Caro and M. Dorigo (1998), *“AntNet: Distributed stigmergetic control for communications networks,*” Journal of Artificial Intelligence Research, vol. 9, pp. 317–365.

F. Ducatelle, G. Di Caro, and L.M. Gambardella (2005), *“Using ant agents to combine reactive and proactive strategies for routing in mobile ad hoc networks,”* International Journal of Computational Intelligence and Applications, vol. 5, no. 2, pp. 169–184.

G. Di Caro, F. Ducatelle, and L.M. Gambardella (2005), *“AntHocNet: An adaptive nature inspired algorithm for routing in mobile ad hoc networks,”* European Transactions on Telecommunications, vol. 16, no. 5, pp. 443–455.

Marco Dorigo, Thomas Stutzle (2001), *“The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*[*]*”*, Technical Report IRIDIA-2000-32.

* To appear in Metaheuristics Handbook, F. Glover and G. Kochenberger (Eds.), International Series in Operations Research and Management Science, Kluwer, 2001.

Zhi-hui Zhan and Jun Zhang (2009), “*Discrete Particle Swarm Optimization for Multiple Destination Routing Problems”*, Springer-Verlag Berlin Heidelberg, EvoWorkshops 2009, LNCS 5484, pp 117–122.

[38] Guohui Zhang et al. (2009), *“An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem”*, Computers & Industrial Engineering 56, pp 1309–1318.

Zhi-Hui Zhan (2009), *“Adaptive Particle Swarm Optimization”*, IEEE Transactions on Systems, Man, and Cybernetics—part b: Cybernetics, vol. 39, no. 6, pp 1362-1381, December 2009.

Yu Bin et al. (2009), *“An improved ant colony optimization for vehicle routing problem”*, European Journal of Operational Research 196 (2009), pp 171–176.