Let the ants deploy your software - An ACO based deployment optimisation strategy

Aldeida Aleti, Lars Grunske, Indika Meedeniya, Irene Moser
Faculty of ICT, Swinburne University of Technology
Hawthorn, VIC 3122, Australia
E-mail: {aaleti,lgrunske,imeedeniya,imoser}@swin.edu.au

***Abstract*—Decisions regarding the mapping of software components to hardware nodes affect the quality of the resulting system. Making these decisions is hard when considering the ever-growing complexity of the search space, as well as conflicting objectives and constraints. An automation of the solution space exploration would help not only to make better decisions but also to reduce the time of this process. In this paper, we propose to employ Ant Colony Optmisation (ACO) as a multi-objective optimisation strategy. The constructive approach is compared to an iterative optimisation procedure - a Genetic Algorithm (GA) adaptation - and was observed to perform suprisingly similar, although not quite on a par with the GA, when validated based on a series of experiments.**

***Keywords*-Ant Colony Optimisation, Component Deployment**

I. Introduction

The quality of software-intensive systems is highly dependent on the design decisions that map software components to hardware hosts [12], [16]. As an example, the deployment of two components implementing two safety-critical functions to the same host may compromise safety due to common cause failures. On the other hand, the performance of the overall system decreases when frequently interacting components are deployed to different hosts. Engineers have to choose from combinatorial design options increasing exponentially with the number of components.

Moreover, the design of software-intensive systems becomes more challenging [20] due to the conflicting nature of the quality requirements such as safety, reliability and performance, just to mention a few. The industry has recognised [3], [20] that automatic methods are needed to support engineers in exploring the design space and finding good solutions. To cope with the conflicting quality requirements multi-objective optimisation strategies are needed. Due to the complexity of the problem, approximate methods are required which find acceptable solutions quickly. These can be grouped into two main categories, namely iterative and constructive algorithms [2].

For the component deployment optimisation mostly iterative methods and specifically evolutionary algorithms (EAs) [5], [16], [17], [18], [19], [22] have been used. EAs are easy to implement and have proved to be robust on complex search spaces [11].

Constructive algorithms have not yet received this degree of attention in the component deployment domain. Although constructive algorithms may stagnate, they often converge quickly and produce diverse solutions [2]. One of the best representatives of constructive algorithms is Ant Colony Optimisation (ACO) [7].

In this paper, we investigate the applicability of ACO-based algorithms to solving component deployment problems with multi-objective optimisation strategies. Consequently, we adapt the Pareto-Ant Colony Optimisation (P-ACO) proposed by Doerner et al. [6] to this optimisation domain and compare it with the Multi-Objective Genetic Algorithm (MOGA). P-ACO is an extension of ACS [4], which is one of the most successful ACO algorithms [23]. To evaluate the performance of the algorithms we use four component deployment problems and run a series of experiments for each of the optimisation algorithms. To compare the results of these experiments we use summary attainment surfaces [14] and the hypervolume indicator [24].

In summary the main contributions of this paper are as follows:

- We formulate software deployment as a multi-objective, multi-constraint optimisation problem and solve it with an adapted ACO optimisation algorithm.
- We compare the performance of P-ACO with MOGA by performing a set of experiments and comparing the summary attainment surfaces and the hypervolume indicators for the obtained pareto fronts.

II. Component Deployment Optimisation Model

The notion of a deployment architecture for an embedded system refers to the allocation of software components to hardware hosts and the assignment of inter-component communications to network links. We model the embedded system as a set of components (software) and a set of hosts (hardware) as listed below.

The decisions regarding the deployment architecture influence not only the functional attributes, but also the quality of the resulting system, which, as pointed out in the literature, e.g. by Papadopoulos and Grante [18], is at least as important as its functionality. The quality of the system is commonly measured in terms of non-functional quality attributes such as safety, reliability, performance and maintainability. The goal is to find the best possible alternatives as quickly as possible.

* Let $C = \{c_1, c_2, ..., c_n\}$, where $n \in \mathbb{N}$, is a set of software components.
* We assume the following parameters for the software architecture:
  - Component (memory) size $size : C \rightarrow \mathbb{N}$.
  - Component communication frequencies $freq : C \times C \rightarrow \mathbb{R}$, where $freq(c_i, c_j) = 0$ if $c_i = c_j$ or there is no communication between c_i and c_j.
  - Component event sizes $evtsize : C \times C \rightarrow \mathbb{N}$, where $evtsize(c_i, c_j) = 0$ if $c_i = c_j$ or there is no event occuring c_i and c_j.
* Let $(H) = \{h_1, h_2, ..., h_m\}$, where $m \in \mathbb{N}$ be a set of hardware hosts.
* We assume the following parameters for the hardware architecture:
  - Host (memory) capacity $cap : H \rightarrow \mathbb{N}$
  - Network bandwidth $bw : H \times H \rightarrow \mathbb{N}$, where $bw(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j.
  - Network reliability $rel : H \times H \rightarrow \mathbb{R}$, where $rel(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j.
  - Network delay $nd : H \times H \rightarrow \mathbb{N}$, where $nd(h_i, h_j) = 0$ if $h_i = h_j$ or there is no network connection between h_i and h_j.

For our approach, two non-functional quality attributes are considered, i.e. Data Transmission Reliability (DTR) and Communication Overhead (CO) as defined by Malek [15]. In an embedded system, sophisticated data recovery mechanisms, like re-transmission, are discouraged. The deployment architecture should be designed such that it minimises the overhead enforced by the data communication for a given set of system parameters. As a network- and deployment-dependent metric, the overall communication overhead of the system is used to quantify this aspect. The deployment problem has therefore been modelled as a biobjective problem.

* Let $f_{DTR} : D \rightarrow \mathbb{R}$ be the Data Transmission Reliability of the system defined as:

$$f_{DTR}(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} freq(c_i, c_j) \cdot rel(d(c_i), d(c_j))$$

* Let $f_{CO} : D \rightarrow \mathbb{R}$ be the Communication Overhead of the system, such that:

$$f_{CO}(d) = \sum_{i=1}^{n} \sum_{j=1}^{n} freq(c_i, c_j) \cdot nd(d(c_i), d(c_j)) + \\ + \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{freq(c_i, c_j) \cdot evtsize(c_i, c_j)}{bw(d(c_i), d(c_j)) \cdot rel(d(c_i), d(c_j))}$$

The deployment problem imposes a number of constraints, such as the localisation constraint which restricts the allocation of a software component to a specific hardware host. Moreover, the co-localisation constraint restricts the allocation of two software components to two different hosts. Finally, the memory constraint checks whether the host has enough memory to perform the tasks of the component.

* Let $\omega_{mem} : D \rightarrow \{0, 1\}$ be the memory constraint of the system defined as:

$$\omega_{mem}(d) = \begin{cases} 0 & \forall h \in H : \sum_{c \in d^{-1}(h)} size(c) \leq cap(h) \\ 1 & otherwise \end{cases}$$

where $d^{-1} : H \rightarrow \mathcal{P}(C)$ is the reverse function of d.
* Let $\omega_{loc} : D \rightarrow \{0, 1\}$ be the localisation constraint defined as:

$$\omega_{loc}(d) = \begin{cases} 0 & \forall c \in C, h \in H : (R_{loc}(c) = h \Rightarrow d(c) = h) \\ 1 & otherwise \end{cases}$$

where R_{loc} is the localisation restriction defined as:

$$R_{loc} : C \rightarrow H \cup \{\perp\}$$

* Let $\omega_{coloc} : D \rightarrow \{0, 1\}$ be the co-localisation constraint defined as:

$$\omega_{loc} : \begin{cases} 0 & \forall c_1, c_2 \in C, c_1 \neq c_2 : \\ & \quad (R_{coloc}(c_1, c_2) = 1 \Rightarrow d(c_1) = d(c_2)) \wedge \\ & \quad (R_{coloc}(c_1, c_2) = -1 \Rightarrow d(c_1) \neq d(c_2)) \\ 1 & otherwise \end{cases}$$

where R_{coloc} is the localisation restriction defined as:

$$R_{coloc} : C \times C \rightarrow \{0, 1\}$$

III. Algorithm Description

In the current work, P-ACO has been applied to the component deployment problem. The implementation of the algorithm for the problem at hand is based on the P-ACO algorithm, which has been adapted to the component deployment problem as described below. To compare the results of this new approach, the Multi-Objective Genetic Algorithm (MOGA) is used, whose main steps are also given in the following subsection.

A. Ant Colony Optimisation

In ACO, ants build a solution by adding randomly selected components and assigning them to hosts. Assignments are made stochastically biased by the pheromone levels on the component/host assignment for each objective. While moving from one component to another, constraints Ω are used to prevent ants from building infeasible solutions.

Ants are prevented from making infeasible assignments. The choice of feasible assignment is made according to the pseudo-random-proportional rule first devised in [8]. Assuming a set of 'feasible' hosts $H = \{h_1, h_2, ..., h_H\}$, a feasible move is made:

$$h = \begin{cases} arg\ max_{i \in H}\{[\sum_{k=1}^{K} (w_k \cdot \tau_i^k)]^\alpha\} & if\ q \leq q_0, \\ \hat{h} & otherwise \end{cases}$$

where q is a uniform random number and q_0 is a parameter set by the user in the interval [0,1]. It determines whether the host with the maximum pheromone value is chosen with a probability of 1. If $q > q_0$, the random-proportional rule is applied, symbolised by $\hat{h}$ in the above equation. $\hat{h}$ is

defined as the probability distribution used for the 'non-greedy' choice, where the host is chosen according to its proportion of pheromone:

$$P_h(x) = \begin{cases} \frac{[\sum_{k=1}^{K}(w_k \cdot \tau_i^k)]^\alpha}{\sum_{c \in C}([\sum_{k=1}^{K}(w_k \cdot \tau_c^k)]^\alpha)} & if\ (c,h) \\ 0 & otherwise \end{cases}$$

Here, as in the previous equation, K is the number of objectives to be optimised. A separate pheromone table is kept for each of the objectives. To determine the total proportion of pheromone per host for the choice of next assignment, weightings w_k are generated randomly for each objective and ant as defined in [13].

The local pheromone update rule decreases the pheromone values with every component/host assignment as follows:

$$\tau_h^k \ = \ (1-\rho) \cdot \tau_h^k + \rho \cdot \tau_0$$

where ρ is the evaporation rate in the interval [0,1]. After m new feasible system deployments, the pheromone tables of each objective are changed according to the global pheromone update rule:

$$\tau_h^k \ = \ (1-\rho) \cdot \tau_h^k + \rho \cdot \Delta\tau_h^k$$

The pheromone information on a component/host assignment pair is increased by a quantity $\Delta\tau_h^k$ if a component/host assignment is part of the best solution according to the respective objective. The global pheromone update is based on the best and second-best deployments for each objective, with half the base pheromone quantity $\Delta\tau_i^k$ allocated on component/host pairs belonging to the second-best solution. The performance of the algorithm has been reported to show little sensitivity to these parameters [6]. During this exploration process, the nondominated solutions are stored in a separate approximation set.

Finally, the new deployment solutions are assessed for dominance of the solutions in the approximation set. If a newly created solution dominates at least one solution in the approximation set, it is added and all deployments dominated by it are removed.

B. Genetic Algorithm

The Genetic Algorithm maintains a population of solutions that evolve simultaneously by means of the genetic operators. For the purpose of the current work, it is meaningful to model the individuals as arrays of component/host assignments. For the deployment problem, the MOGA individual consists of a gene array of length $|C|$, where each gene specifies the host the respective component has been assigned to.

The MOGA starts with a set of n randomly selected system deployments as initial population. After the initialisation, the single-point crossover and mutation operators are applied according to predefined crossover and mutation rates. The mutation operator changes two random assignments by swapping the hosts between two components.

Feasible individuals are added to the population. The selection operator removes as many individuals as needed to maintain the prescribed population size n by eliminating the worst-performing individuals. To determine the performance, the new solutions are evaluated applying the proportional selection strategy defined by Fonseca and Fleming [10], considering each objective function in turn.

The new population is subsequently screened for individuals which dominate solutions in the approximation set. These are added, and the dominated solutions are removed from the approximation set.

IV. Validation

A. Experimental set

The performance of the P-ACO and MOGA algorithms was tested using the Archeopterix [1] tool on four representative datasets, as detailed in Table I. To obtain a fair comparison between different algorithms, the generally accepted approach is to allow the same number of function evaluations for each trial [21]. The test instances have been generated according to the problem descriptions formalised by Mikic-Rakic et al. [16] as well as Malek [15].

Table I
Test datasets

Dataset	1	2	3	4
Hosts – Components	15 – 34	20 – 45	25 – 54	40 – 60

B. Algorithmic settings and parameters

P-ACO uses $m = 10$ ants for each iteration, after which the pheromone map is updated. It uses a local pheromone evaporation rate of $\rho = 0.1$ and a 40% greedy choice of next assignment ($q_0 = 0.4$), as well as an increase of $\Delta\tau_h^k = 10$ for the best, $\Delta\tau_h^k = 5$ for the second-best solution. The initial pheromone values are $\tau_0 = 1$.

The MOGA implementation uses a population size of 2000 deployments ($n = 2000$), which are initally created uniformly randomly. The mutation and crossover rates are set to 40% with elitism in the sense that we do not replace fitter solutions in the population.

C. Experimental Results

As stochastic solvers do not provide solutions predictably, results concerning the performance of stochastic solvers are usually reported as mean values over repeated trials. For the current comparison, all algorithms were granted 25 000 function evaluations per trial with 50 repeats.

Averaging over multiple pareto fronts is not a meaningful reporting method. Researchers commonly use the 'summary attainment surface', the hypersurface created when connecting the $n-th$ intersections of the plots, as developed by Knowles [14] on the basis of seminal work by Fonseca and Fleming [9].

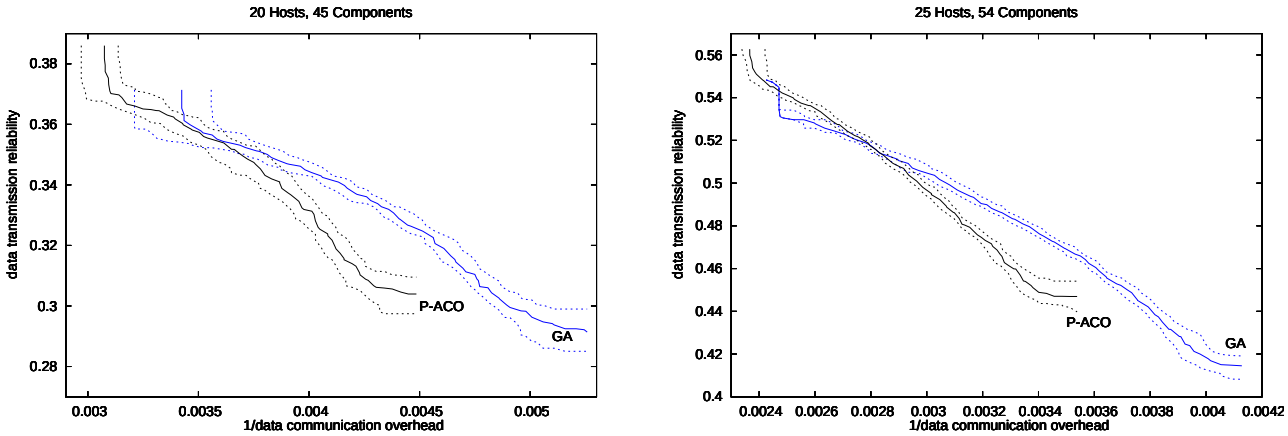

Figure 1. Summary attainment surfaces for 2 problem instances (datasets). The solid lines represent the 'mean' (intersection of the 25th of 50 plots), the dotted lines are the quartiles (intersection of 12th and 38th line respectively). The goal is to maximise the data transmission reliability (y axis) while minimising the communication overhead. For legibility, the communication overhead has been inversed (x axis), so that both objectives can be maximised.

Figure 1 shows the summary attainment surfaces for two of the four datasets in two dimensions. Looking at the 50% summary attainment lines of both algorithms it is difficult to decide which algorithm performs better. P-ACO seems to improve predominantly the data transmission reliability, whereas the GA seems to work more on the data communication overhead, a possible indication that the two algorithms might complement each other.

Other metrics for comparing the performance of multi-objective optimisers exist. Zitzler et al. [24] recommend the hypervolume indicator. It measures the hypervolume between the approximation set surface and a fixed point in the result space. The most intuitive reference point is zero in each dimension, which is the value used in our reporting. The 50 trial outcomes for each algorithm were used to establish the hypervolume (a simple surface in two dimensions) between (0.0, 0.0) and the objective values of the non-dominated set. The average, minimum and maximum values are listed in Table II.

The data shows that MOGA outperforms P-ACO consistently on all aggregates of the hypervolume indicator. In the hypervolume development graphs, examplified in Figure 2 by values from one of the four datasets, P-ACO improves more rapidly than MOGA during the first iterations, after which it stagnates. MOGA continues to improve throughout the remainder of the trials.

Table II
HYPERVOLUME INDICATORS FOR ALGORITHMS MOGA AND P-ACO. HIGHER VALUES INDICATE BETTER PERFORMANCE.

		15h/34c	20h/45c	25h/54c	30h/60c
MOGA	min	0.0048	0.0016	0.0020	0.0015
	max	0.0054	0.0018	0.0021	0.0017
	mean	0.0051	0.0017	0.0020	0.0016
P-ACO	min	0.0043	0.0014	0.0017	0.0014
	max	0.0050	0.0016	0.0018	0.0016
	mean	0.0046	0.0015	0.0018	0.0015

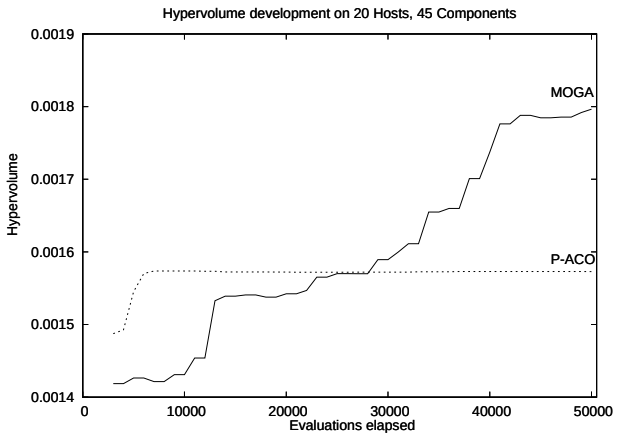

Figure 2. Development of the hypervolume as the algorithms progress.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have investigated the use of P-ACO [6] for finding solutions to multi-objective multi-constraint component deployment problems and compared it with a MOGA application to the same problem instances. ACO-based algorithms are constructive optimisation algorithms and can be expected to work well with highly constrained problems. The P-ACO implementation finds a diverse set of solutions quickly. However, in the long run the optimisation progress stagnates. The use of a local search in combination with ACO has been recommended [8] in order to address this issue. In our future work we are therefore interested to combine the qualities of the two algorithms in a hybrid approach, which has showed promise in preliminary work. We further plan to integrate additional quality attribute eval-

uation models. For the comparison and the use of additional evaluation models our tool Archeopterix [1] will be used as the experimental platform.

REFERENCES

[1] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya, "Acheopterix: An extendable tool for architecture optimisation of aadl models," in *MOMPES'09*. IEEE Digital Libraries, 2009, pp. 61–71.

[2] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv*, vol. 35, no. 3, pp. 268–308, 2003.

[3] M. Broy, "Challenges in automotive software engineering," in *(ICSE'06)*. ACM, 2006, pp. 33–42.

[4] O. Cordon, F. Herrera, and T. Stützle, "A review on the ant colony optimization metaheuristic: Basis, models and new trends," *Mathware and Soft Comp.*, vol. 9, pp. 141–175, 2002.

[5] R. P. Dick and N. K. Jha, "MOGAC:A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Hierarchical Heterogeneous Distributed Embedded Systems," *IEEE Tran. on Cyclic Analog-to-Digital Converter Synthesis*, vol. 17, no. 10, pp. 920–935, 1998.

[6] K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer, "Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection," *Annals of Op. Res.*, vol. 131, no. 1–4, pp. 79–99, 2004.

[7] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *TCS: Theoretical Computer Science*, vol. 345, pp. 243–278, 2005.

[8] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," in *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, April 1997, pp. 53–66.

[9] C. M. Fonseca and P. J. Fleming, "On the performance assessment and comparison of stochastic multiobjective optimizers," in *Parallel Problem Solving from Nature, PPSN IV*. Springer, 1996, pp. 584–593.

[10] C. Fonseca and P. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," in *Proc. of the Fifth International Conference on Genetic Algorithms*, 1993, pp. 416–423.

[11] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.

[12] J. Fredriksson, K. Sandström, and M. Åkerholm, "Optimizing resource usage in component-based real-time systems," in *CBSE'05*, ser. LNCS, vol. 3489. Springer, pp. 49–65.

[13] S. Iredi, D. Merkle, and M. Middendorf, "Bi-criterion optimization with multi colony ant algorithms," *Lecture Notes in Computer Science*, vol. 1993, pp. 359–372, 2001.

[14] J. Knowles, "A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers," in *Intelligent Systems Design and Applications*, 2005, pp. 552–557.

[15] S. Malek, "A user-centric approach for improving a distributed software systems deployment architecture," PhD in Computer Science, Faculty of The Graduate School University of Southern California, 2007.

[16] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "A decentralized redeployment algorithm for improving the availability of distributed systems," in *Component Deployment*, ser. LNCS, vol. 3798. Springer, 2005, pp. 99–114.

[17] M. Nicholson, "Selecting a topology for safety-critical real-time control systems," Ph.D. dissertation, Department of Computer Science, University of York, 1998.

[18] Y. Papadopoulos and C. Grante, "Evolving car designs using model-based automated safety analysis and optimisation techniques," *The Journal of Systems and Software*, vol. 76, no. 1, pp. 77–89, 2005.

[19] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transaction Computers*, vol. 55, no. 2, pp. 99–112, 2006.

[20] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *ISCE'07*, 2007, pp. 55–71.

[21] R. L. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial," *Journal of Heuristics*, vol. 7, no. 3, pp. 261–304, 2001.

[22] D. Salazar, C. M. Rocco, and B. J. Galvan, "Optimization of constrained multiple-objective reliability problems using evolutionary algorithms," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 1057–1070, 2006.

[23] T. Stützle and M. Dorigo, "Controlled experimental design for statistical comparison of integer programming algorithms," *IEEE-EC*, vol. 6, pp. 358–365, 2002.

[24] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance Assessment of Multiobjective Optimizers: An Analysis and Review," vol. 7, no. 2, 2003, pp. 117–132.