Multi-Objective Optimization of Software Architectures
Using Ant Colony Optimization

Carsten Mueller, Member, IACSIT

Abstract—In current practice, software architects try to find good solutions manually, which is time-consuming, error-prone and often lead to suboptimal designs. Architecture trade-off analysis methods are appropriate techniques to evaluate design decisions and design alternatives with respect to conflicting quality requirements. In this research article an automated approach guided by ant swarm intelligence to search the design space for optimal solutions is proposed. This approach applied multi-objective ant colony optimization to the trade-off problem in the context of software architectures.

Index Terms—Trade-off, software architecture, multi-objective optimization, ant colony optimization.

I. INTRODUCTION

A. Software Architecture Design and Trade-off

The development of software-intensive technical systems is challenging, because software and system engineers have to deal with a large number of non-functional or quality requirements such as safety, availability, reliability, maintainability and temporal correctness requirements [1]. One major difficulty is that these non-functional requirements conflict with one another and with economic constraints. To construct a system that fulfils all its quality requirements is often not possible. As a consequence, system engineers have to consider several design alternatives and identify a solution that fulfils most quality objectives.

Software architecture design is influenced to a large extent by the consideration of quality attributes, such as performance, reliability, maintainability, costs, or security. Experienced software architects intuitively know styles and tactics to improve quality attributes of a software architecture [2]. In recent years, many researchers have proposed to encode architectural design decisions into software architecture models (e.g., using architecture description languages or UML) [3] thus enabling automated reasoning.

For the evaluation of software architectures, many specific approaches for the different non-functional properties are described in the literature [4]-[7]. Many real-world optimization problems in the context of software architectures have multiple and conflicting objectives. As a result, one single solution for these problems often doesn't exist and traditional optimization techniques are not suitable. To deal with multi-objective optimization (MOO) problems several techniques have been developed. The aim of these technique is to find a set of solutions, where none of the solutions of this set is superior to the others. These solutions are known as Pareto-optimal solutions.

B. Partitioning of a Software Architecture

The partitioning of a software architecture into a set of units is generalized to the well-known set partitioning problem [8]. A partition of a set \( S \) is a collection of disjoint subsets of \( S \) whose union is \( S \).

For example, there are five alternatives to partition the set \( \{a, b, c\} \). These alternatives are: \( \{\{a\}, \{b, c\}\}, \{\{a\}, \{b\}, \{c\}\}, \{\{a\}, \{b\}, \{c\}\}, \{\{a\}, \{b\}, \{c\}\}, \{\{a\}, \{b\}, \{c\}\} \).

The number of ways to partition a set of \( n \) elements into \( k \) nonempty subsets is computed by the Stirling numbers of the second kind, \( S(n, k) \) [8].

It is calculated with the following recursive formula.

\[
S(n, k)=k \times S(n-1, k) + S(n-1, k-1), \quad n \geq 1
\]

The total number of ways to partition a set of \( n \) elements into arbitrary number of nonempty sets is counted by the \( n \)th Bell number as follows [8].

\[
B_n = \sum_{k=1}^{n} S(n, k)
\]

In theory, \( B_n \) is the total number of partitions of a software architecture with \( n \) components. \( B_n \) grows exponentially with \( n \). For example, \( B_1 = 1 \), \( B_3 = 5 \), \( B_5 = 52 \), \( B_7 = 877 \), \( B_{15} = 1.382.958.545 \). Therefore, searching for a feasible design alternative for the software architecture becomes problematic as \( n \) (i.e. the number of components) grows.

C. Multi-Objective Optimization

In MOO, solutions are related according to the concept of dominance which provides MOO practitioners a means by which multiple solutions are compared and subsequently ranked or sorted. A standard dominance measure compares two solutions \( (s_1, s_2) \) and determines which of the following conditions and definitions apply:

- If \( s_1 \) is better in all objectives than \( s_2 \). \( s_1 \) is said to strongly dominate \( s_2 \), denoted as \( s_1 \gg s_2 \).
- If \( s_1 \) is not worse than \( s_2 \) in all objectives and better in at least one objective, \( s_1 \) is said to dominate \( s_2 \), denoted as \( s_1 \succ s_2 \).
- If \( s_1 \) is not worse than \( s_2 \) in all objectives, \( s_1 \) is said to weakly dominate \( s_2 \), denoted as \( s_1 \succeq s_2 \).
• If \( s_i \) does not weakly dominate \( s_2 \), nor \( s_i \) weakly dominate \( s_2 \), and \( s_1 \) and \( s_2 \) are incomparable, \( s_1 \not\equiv s_2 \).

A non-dominated set is a set of solutions that are not weakly dominated by any other solution in the set. The Pareto optimal set is a set of optimal non-dominated solutions.

A multi-objective optimization problem is defined as follows: Find a solution \( x \), which is an element of the solution space \( X \), satisfies a set of constraints \( g_i(x) \) and weakly evaluate a specific \( f(x) \), whose elements represent the objective function \([9], [10]\).

If this definition is applied to the quality optimization problem in the architecture design phase, the solution space \( X \) describes all possible software architecture specifications that fulfill the functional requirements. A constraint \( g_i(x) \) could describe budget limitations, technical feasibility constraints or a certain threshold for non-functional properties. A function \( f_i(x) \) of \( f(x) \) evaluates a specific quality (e.g. reliability) of the software architecture \( x \).

Each decomposition alternative in the large design space have both pros and cons. The selection of a particular decomposition among the feasible alternatives leads to a trade-off.

II. ANT COLONY OPTIMISATION

Ant Colony Optimization (ACO) [11] is a metaheuristic inspired by the indirect communication of real ants by means of trails of a chemical substance called pheromone. Artificial ants are simple agents that use numerical information (artificial pheromone information) to communicate their experience while solving a particular problem to other ants. These principles provide a common framework for most applications of ant algorithms to combinatorial optimization problems. Algorithms derived from the ACO metaheuristic are called ACO algorithms.

In ACO algorithms, an individual ant constructs candidate solutions to a combinatorial optimization problem by starting with an empty solution and then iteratively adding solution components until a complete candidate solution is generated. The ants make use of information that reflects the experience accumulated by previous ants, called pheromone information, and of problem dependent information, called heuristic information, in order to decide which solution component will be added to its current partial solution by means of a stochastic construction policy [12]. Every step in an ACO algorithm at which all ants complete a feasible solution will be called iteration. Each ant represents unequivocally a specific solution during a certain iteration.

ACO algorithms have been shown to be effective problem solving strategies for a wide range of problem domains, including Multiple Objective Optimisation (MOO).

An important step when designing an ACO algorithm for a specific problem is the definition of the pheromone and heuristic information. The pheromone update strategy decides which ants are allowed to modify the pheromone information and how the selected ants modifies it. Solution components which are part of better solutions or are used by many ants will receive a higher amount of pheromone. To prevent unlimited accumulation of the pheromone trail before the pheromone information is updated all pheromone trails are decreased by a factor that models evaporation of the trails. The pheromone trail associated with a solution component decreases exponentially if it does not receive any amount of pheromone. Evaporation enables the algorithm to forget bad choices over time. Selected solutions are improved local search methods.

A general algorithmic scheme is followed. Parameters and pheromone information are initialized. A main loop is repeated until a termination condition is met, which is a given number of evaluated solution constructions or a limit of the available computation time. In this main loop, ants construct feasible solutions by adding solutions components. Additionally, these solutions are improved using a local search. Next, the new best solution found (best-so-far solution) is determined. A number of solutions which include the best-so-far solutions are selected to update the pheromone information. Before updating the pheromone information, evaporation is performed by decreasing the pheromone trail by a factor \( p \). Finally, when the main loop ends, the algorithm returns the best solution found since the start of the algorithm.

III. HYBRID APPROACH

A. Overview

This approach starts considering as input an initial software architecture model of the system, named initial candidate.

The optimization process starts with the search problem formulation. In this step, three degree of freedom types are considered: 1) allocation of components, 2) server configuration and 3) component selection. The result of this step is a set of degree of freedom instances that describe the search problem.

In the second step, the search problem is optimized using analytical techniques with simplified QoS evaluation and a simplified search space formulation. The impact of a degree of freedom (or a combination of them) is evaluated by analytical methods, and the Pareto optimal candidates for the simplified search space are derived efficiently by solving a mixed integer linear programming problem. The result of this step is a set of candidates that are Pareto optimal with respect to the simplified QoS evaluation and the simplified search space.

In the third step, the results of the analytical optimization are used as input candidates for the ant colony optimization of the original search problem. The result of this phase is a set of near Pareto optimal candidates. This set is presented to the software architect, who studies the remaining optimal trade-offs between possibly conflicting objectives.

B. Search Problem Formulation

A candidate architecture is Pareto optimal, if it is superior to all other candidate in at least one quality criterion. Let \( a \) be a candidate solution, let \( SC \) be the set of all possible candidates, and let \( q \) be a quality criterion with a value set \( C_q \).
an evaluation function \( f_q : SC \rightarrow C \), so that \( f_q(c) \) denotes the quality property of a \( c \in SC \) for the quality criterion \( q \). and an order \( \leq q \) on \( D_q \) that \( c_1 \leq q c_2 \) means that \( c_1 \) is better than or equal to \( c_2 \) with respect to quality criterion \( q \). A candidate solution \( a \) is Pareto optimal if it is better than any other candidate \( a^* \) in at least one quality criterion or equally good than \( b \) in all criteria, i.e. iff \( \forall b \in SC(\exists q: f_q(a) \leq q f_q(b) \lor \forall q: f_q(a) = q f_q(b)) \)

If a candidate solution is not Pareto optimal, then it is Pareto-dominated by at least one other candidate solution in \( SC \) that is better or equal in all quality criteria.

The optimization problem is formulated for a set of quality criteria \( Q = \{q_1, \ldots, q_m\} \). \( \min_{c \in SC} \{f_{q_1}(c), \ldots, f_{q_m}(c)\} \) In this hybrid approach three quality criteria are considered: \( q_1 = T = \) mean response time, \( q_2 = A = \) availability measured as the probability of success of each request, and \( q_3 = C = \) cost.

A candidate \( c \) is part of the Pareto front with respect to a set of candidate solutions \( S \subseteq SC \) if it is not Pareto-dominated by any other candidate from \( S \), i.e. iff \( \forall b \in S(\exists q: f_q(c) \leq q f_q(b) \lor \forall q: f_q(c) = q f_q(b)) \). An optimization problem with \( Q \) objectives is assumed. An Ant \( k \) adds \( j \) to its partial solution with the following probability:

\[
P_{Sj}^k = \frac{\prod_{q=1}^{m} (\tau_{ij}^q)^{\alpha} \cdot \prod_{q=1}^{m} (\eta_{ij}^q)^{\beta}}{\sum_{l \in N_j^k} \left[\prod_{q=1}^{m} (\tau_{ij}^q)^{\alpha} \cdot \prod_{q=1}^{m} (\eta_{ij}^q)^{\beta}\right]} \tag{3}
\]

if \( j \in N_S^k \) where \( S = \{S_1, \ldots, S_q\} \) is the state vector of the current partial solution and each element \( S_q \) of this vector is value needed by each objective \( q = 1, \ldots, Q \) to define a solution component; \( N_k^j \) is the feasible neighbourhood of ant \( k \) given the current state vector \( S \); \( \tau_{ij}^q \) is the pheromone information of \( j \) given \( S_q \) for the \( q^{th} \) objective in the current iteration and, in an equivalent way, \( \eta_{ij}^q \) is the heuristic information of \( j \) given \( S_q \) for the \( q^{th} \) objective and it is seen as the \( q^{th} \) component of a weight vector \( \lambda \) taken from set \( \Lambda \).

C. Solution construction

For the software architecture trade-off the goal is finding a path which has the optimal sum of QoS value from the start point to target point in the weighted directed acyclic graph.

The appointed start point is set as the ants nest and target point is set as the food source. Finding the optimal sum of QoS value is abstracted to finding the path from the ants nest to the food source.

Solution construction is biased by heuristic and pheromone values which are combined using a specific weighting rule. The solution construction component type is distinguished by the specific combination of heuristic and pheromone information. These combinations are:

- **Targeted**: Information (pheromone or heuristic) pertaining to only a specific objective is used in the solution construction process.
- **Dynamic**: This information that spans more than one objective (multiple heuristics / pheromone matrices) is used in the solution construction process.

The pheromone trail update and decay method determines what information is used in the update process and which elements are affected.

ACO algorithms for single objective problems represent the pheromone information by a pheromone matrix (or vector) where each entry in the matrix corresponds to the desirability for a certain solution component.

The pheromone matrix is chosen between two standard models: single and multiple.

A **single pheromone matrix** - used in traditional single objective ACO algorithms - is used where each decision variable corresponds to one pheromone value.

**Multiple pheromone information and matrix** are used when different pheromone information is defined for each objective and weights are used to aggregate them into a single value, in a similar way that weights are used to aggregate different objectives in the weighted sum scalarization [13] of a multi-objective problem.

Each pheromone matrix contains different pheromone values depending on the specific implementation. If a one-to-one pheromone to objective mapping is used then ideally the pheromone information contained in a single pheromone matrix will reflect which solution. One ant select a set of iteration-best or best-so-far-solutions to update the pheromone matrices, with respect to each objective (best-of-objects).

Another way to update the pheromone matrices is to collect and store the non-dominated solutions in an external set. Only the solutions in the non-dominated set are allowed to update the pheromone matrices.

\( \tau_q \) denotes the initial pheromone concentration value on each path and \( \tau_q(t) \) the pheromone concentration value on path \((i,j)\) at time \( t \). The pheromone concentration on the path \((i,j)\) at time \((t+1)\) is expressed as

\[
\tau_q(t+1) = \rho \cdot \tau_q(t) + \sum_{k=1}^{l} \Delta \tau_q^k
\tag{4}
\]

where constant \( \rho \) stands for the volatile coefficient of pheromone between 0 and 1. \( \Delta \tau_q^k \) stands for the pheromone concentration increment between time \( t \) and time \( (t+1) \) on path \((i,j)\) for the \( k^{th} \) ant and \( l \) stands for the number of ants.

\( \Delta \tau_q^k \) is directly proportional to QoS in software architecture design and trade-off. In order to utilize the global information (multiple pheromone information and matrix) and guarantee limitation of residual pheromone accumulation,
the ant quality model is employed. Pheromone concentration increment is denoted by \( \Delta \tau_{ij}^k(t, t+1) = f(t) / C \) where \( C \) is a constant, \( f(t) \) stands for the QoS measurement index of selected modules for current evaluated software architecture, which is expressed as

\[
f(t) = \sum_{j=1}^{h} \sum_{j=1}^{m} w_j Q_{ij} \tag{5}
\]

where \( w_j \) denotes the weight of the \( j \)-th QoS attribute for one module, \( 0 \leq w_j \leq 1 \), \( \sum_{j=1}^{h} w_j = 1 \), \( Q_{ij} \) denotes the \( i \)-th QoS attribute value of the module selected by the \( j \)-th evaluation for software architecture. \( h \) stands for the number of QoS attributes and \( m \) for the number of modules selected in the current evaluated software architecture.

The selection rule of next module node \( u \) at the time \( t \) for the \( k \)-th ant is expressed as

\[
s = \max \left\{ \tau_{iu}(t) \right\}^{q} \left[ \eta_{iu}(t) \right]^{q} \bigg| u \in \text{allowed}_k(i), q \leq q_0 \tag{6}
\]

IV. CONCLUSION

Solving the software architecture trade-off done manually is time-consuming. In this research article an automated approach guided by swam intelligence to search the design space for optimal solutions is proposed. The hybrid approach based on Ant Colony Optimization presented in this paper solves this Multi-Objective Optimization. The ants are able to evaluate a candidate architecture based on QoS and Pareto optimality. This research paper is the first step of a journey and this approach needs intensive further research and evaluation. Updates are found on http://www.itg-research.net.

REFERENCES


Carsten Mueller holds a M.Sc. degree from the University of Liechtenstein in Vaduz and a Ph.D. degree from the University of Economics (Department Informatics and Statistics) in Prague. He works more than 15 years as an independent consultant and trainer in the areas of IT process optimization, Object-oriented Modelling and IT Governance. He is currently the technical head of Processes and IT Management at a large airport in Germany and senior lecturer for software engineering, UML and algorithms at the Baden-Wuerttemberg Cooperative State University.