JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0000-0000 | EISSN 0000-0000 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

A Mixed Integer Linear Programming Model for Reliability Optimisation in the Component Deployment Problem

Asef Nazari, Dhananjay Thiruvady
CSIRO Computational Informatics, Victoria 3169, Australia, asef.nazari@csiro.au, dhananjay.thiruvady@csiro.au

Aldeida Aleti
Faculty of Information Technology, Monash University, Australia
aldeida.aleti@monash.edu

I. Moser
Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia
imoser@swin.edu.au

Component deployment is a combinatorial optimisation problem in software engineering that aims at finding the best allocation of software components to hardware resources in order to optimise quality attributes, such as reliability. The problem is often constrained due to the limited hardware resources, and the communication network, which may connect only certain resources. Due to the non-linear nature of the reliability function, current optimisation methods have focused mainly on heuristic or metaheuristic algorithms. These are approximate methods, which find near-optimal solutions in a reasonable amount of time. In this paper, we present a mixed integer linear programming (MILP) formulation of the component deployment problem. We design a set of experiments where we compare the MILP solver to methods previously used to solve this problem. Results show that the MILP solver is efficient in finding feasible solutions even where other methods fail, or prove infeasibility where feasible solutions do not exist.

Key words: Reliability, integer programming, search, Linear programming

1. Introduction

Finding the best way to allocate software components to hardware resources is a complex software engineering problem. Objective functions are often non-linear, e.g. reliability (Fredriksson et al. 2005), and the problem is highly constrained, with limited available memory, restricted availability of communication networks, and colocalisation restrictions (Aleti et al. 2013). For instance, memory constraints restrict the allocation of software components to hardware resources in such a way that the sum of memory requirements for software components should not exceed the available capacity in the hardware resources. Software systems are usually large. A typical system may consists of more than 30 hardware resources, and more than 400 software functions, which creates

30^{400} possible configurations. The large number of variants has to be handled technically, a task that can take years for a software engineer.

To handle the complex task of designing software systems, recent research has focused on automating the design of software systems using stochastic search methods, such as genetic algorithms (GA) and ant colony system (ACS) (Aleti et al. 2013). These methods operate at the design phase of system implementation, and search for an optimal software architecture that maximises or minimises the quality of the system, and satisfies given constraints. The area is broadly known as search based software engineering (SBSE) (Harman 2007a). SBSE can be applied to different stages of the development of software systems, such as requirements engineering (Zhang et al. 2008), predictive modelling (Afzal and Torkar 2011), design (Aleti et al. 2013), program comprehension (Harman 2007b), and software testing (McMinn 2004, Mantere and Alander 2005, Aleti and Grunske 2015).

The component deployment problem has previously been solved with a population-based ant colony optimisation (P-ACO) (Guntsch and Middendorf 2003) in the multiobjective form, which was found to outperform NSGA-II (Moser and Montgomery 2011) when combined with a suitable local search method. P-ACO is essentially a multiobjective extension of ACS, one of the most successful ACO applications, which is used as a benchmark method in this work. Some other approaches focus on the satisfaction of user requirements (Martens and Koziolek 2009, Calinescu and Kwiatkowska 2009), software quality attributes (Medvidovic and Malek 2007, Sharma et al. 2005, Fredriksson et al. 2005), or both (Mikic-Rakic et al. 2004, Medvidovic and Malek 2007, Fredriksson et al. 2005, Lukasiewycz et al. 2008).

Thiruvady et al. (2014) applied constraint programming coupled with ant colony system (CP-ACS) to optimise the component deployment problem. The work introduced a novel way to deal with constraints, focusing on memory, colocation and communication constraints. Memory constraint restricted the allocation of software components to hardware units based on memory requirements. Colocation constraint enforced the deployment of redundant components into different hardware hosts for safety reasons. Finally, communication constraint checked whether interacting components were allocated into communicating hardware units. Despite the difficulty of constraints, ACS was the most effective method, winning over the CP-ACS method.

Stochastic search finds approximate results where exact approaches cannot be devised and optimal solutions are hard to find. Search methods can be used by practitioners in a fairly 'black box' way without mastering advanced theory, whereas more sophisticated solvers (e.g. mixed integer linear programming solvers) are tailored to the specific mathematical structure of the problem at hand, and can become completely inapplicable if small aspects of the problem change. On the other hand, the applicability of stochastic search methods to highly constrained problems is limited, due to the computational cost in initialising the search with feasible solutions. This has motivated

our work in investigating the applicabitily of mixed integer linear programming solvers to the component deployment problem.

Several mixed integer linear programming formulations have been applied to the component deployment problem (Bowen et al. 1992, Ernst et al. 2003, 2006, Hadj-Alouane et al. 1999), however none of these consider reliability. Reliability is one of the most crucial quality attributes in safety-critical software systems, such as the ones used in the automotive domain. It is defined as a function of failure rates of software components, and is inherently non-linear. In this paper, we present a mixed integer linear programming (MILP) model of the reliability optimisation in the component deployment problem and test its efficiency. The results from the MILP optimisation are compared with results from ant colony optimisation, and the CP-ACS, on a set of experiments designed with problems of different levels of constraint difficulty, and problem instance size.

2. Previous Work

The deployment of software components to distributed hosts presents itself as an optimisation problem in many domains. In many cases, the reliability of the resulting system is one of the core criteria for the success of the software allocations. Dimov and Punnekkat (2005) provide a comprehensive overview of reliability models in software component deployments. Most models consider foremost the reliabilities of the components themselves (Cheung 1980, Krishnamurthy and Mathur 1997, Reussner et al. 2003), which some authors derive from the usage profile (Hamlet et al. 2001, Reussner et al. 2003), while Gokhale et al. (1996) infers it from the test coverage of the components. Most models also include the reliability of component calls (Krishnamurthy and Mathur 1997) or the connections between the components (Singh et al. 2001).

Heydarnoori and Mavaddat (2006) formulated the problem of deploying software in a distributed system as a multi-way cut problem, in which system reliability depends on the reliability of the network between the communicating components. Their approach assigns components deterministically to hosts according to the connectivity the host provides, obtaining a polynomial-time approximation of the optimum.

The majority of reliability studies in distributed systems view the deployment of software as temporary task allocation rather than permanent deployment. Assayad et al. (2004) describe a scheduling problem in which reliability refers to the probability of no component failing during the execution of the workflow. Their biobjective formulation weighs reliability against total runtime (makespan). Similarly, Kartik and Murthy (1997) presented a deterministic algorithm which assigns tasks to heterogeneous processors according to the most-constrained-first principle: tasks which communicate frequently are assigned first. Several MILP formulations have been applied to the

task allocation problem (Bowen et al. 1992, Ernst et al. 2003, 2006, Hadj-Alouane et al. 1999), however none of these consider reliability.

Hadj-Alouane et al. (1999) is arguable the only work that focuses on the task allocation problem in the automotive domain. The goal was to minimise the cost that arises from installing microcomputers and high-speed or low-speed communication links between those. The quadratic 0-1 MILP formulation was compared to a generic algorithm adaptation, which outperformed the MILP by approximately 4% in the quality of the solutions, while using only a fraction of the CPU time (less than one-twentieth) required by the MILP. The data sets used contained 20 tasks and 6 processors and 40 tasks and 12 processors respectively.

Harris (2013) has recently provided a comprehensive overview of the usage of electronics in modern vehicles. The author illustrates how the electronic content of cars has increased over the last 20 years, facilitated by a shift from dedicated devices and wires to ECUs and Ethernet cables. Climate control, ABS and speed control systems have become standard equipment in contemporary cars. In 2002, Leen (2002) pointed out that 23% of the cost of a new car stems from electronics, and 80% of all innovations are software related.

Despite this strong motivation, relatively few researchers have addressed the problem of component deployment in automotive systems. In a comprehensive approach by Papadopoulos and Grante (2005), the functionalities to include in a new vehicle model were first selected with the use of an evolutionary algorithm. The functionalities were then implemented as software components considering profit and cost. The subsequent deployment to hardware modules was optimised with the help of an evolutionary algorithm, with the optimisation focussing on reliability.

Aleti et al. (2009) formulated component deployment as a biobjective problem where data transmission reliability and communication overhead were optimised. Due to the multitude of constraints - memory capacity, location and colocation constraints were considered - a constructive heuristic was chosen (Guntsch and Middendorf 2003, P-ACO) in addition to an evolutionary algorithms (Fonseca et al. 1993, MOGA). P-ACO was found to produce better solutions in the initial optimisation stages, whereas MOGA continued to produce improved solutions long after P-ACO had stagnated.

Using incremental heuristics on such a constrained problem requires constraint handling procedures to overcome infeasible space. Moser and Mostaghim (2010) found that repairing infeasible solutions produced better results than eliminating or penalising solutions evolved by NSGA-II.

In the work by Aleti and Meedeniya (2011), a Bayesian approach was adapted to the formulation of (Aleti et al. 2009), and found to outperform NSGA-II (Deb et al. 2000) as well as P-ACO on the hypervolume indicator. A formulation by Meedeniya et al. (2012) treated response time and reliability as probability distributions and presented solutions which are robust with regards to

the uncertainty. Given different external factors, invocations of components can lead to different effects.

Component allocation in distributed systems is often explored from the point of view of redundancy allocation, which was studied by Aleti (2015), who considered response time, reliability and cost of the system in a triobjective formulation optimised using GA. Sheikhalishahi et al. (2013) applied a hybrid between a genetic algorithm and particle swarm optimisation (PSO) to a multiobjective problem which minimises cost, system volume and weight while maximising system reliability. The GA optimises the number of redundancies for a subsystem, whereas the PSO optimises the reliability of these redundancies a GA solution at a time. Liang and Smith (1999) adapted Ant System to the redundancy allocation problem, adding a mutation operator instead of a local search phase and a penalty function to discourage constraint violations. Kumar et al. (2009) argued that redundancy should not be applied at the component level alone, but also consider the redundancy of modules. Using a hierarchical GA to this formulation, the authors conclude that optimising redundancy on multiple levels yields more reliable configurations.

3. The Component Deployment Problem

The architecture of an embedded system represents a model or an abstraction of the real elements of the system, such as software components, hardware units, interactions of software components, communications between hardware units, and their properties. The set of software components $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ is annotated with the following properties:

(i) sz_i: memory size of component c_i, expressed in KB (kilobytes).

(ii) wl_i: computational requirement of component c_i, expressed in MI (million instructions).

(iii) q_i: the probability that the execution of a system starts from component c_i.

The execution of the software system is initiated in one software component with a given probability q_i. During the execution of the software system many other components are activated through interactions, which are assigned transition probabilities (Kubat 1989). Software interactions are specified for each link from component c_i to c_j with the following properties:

(i) ds_{ij}: the amount of data sent from software component c_i to c_j during a single communication event, expressed in KB (kilobytes).

(ii) p_{ij}: the probability that the execution of component c_i ends with a call to component c_j.

Software components and their interactions are illustrated in fig. 1.

The hardware architecture is composed of a distributed set of hardware hosts, denoted as $\mathcal{H} = \{h_1, h_2, ..., h_m\}$, where $m \in \mathbb{N}$, with different memory capacities, processing power, access to sensors and other peripherals. Each hardware host has the following properties:

(i) cp_i: memory capacity of the hardware host h_i, expressed in KB (kilobytes).

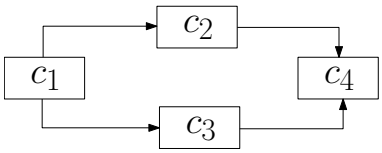

Figure 1 Software components and interactions.

(ii) ps_i: the instruction-processing capacity of hardware unit h_i, expressed in MILPS (million instructions per second) (Feiler and Rugina 2007).

(iii) fr_i: characterises the probability of a single hardware unit failure (Assayad et al. 2004).

The hardware hosts are connected via network links, which are used by interacting components, and denoted as $\mathcal{N} = \{n_1, n_2, ...n_s\}$. Network links have the following properties:

(i) dr_{ij}: the data transmission rate of the bus that connect hardware units h_i, h_j, expressed in KBPS (kilobytes per second).

(ii) fr_{ij}: failure rate of the bus that connects hardware units h_i, h_j characterizes the probability network failing to transmit the data.

The architecture of a system with three hardware hosts and one network link is shown in fig. 2.

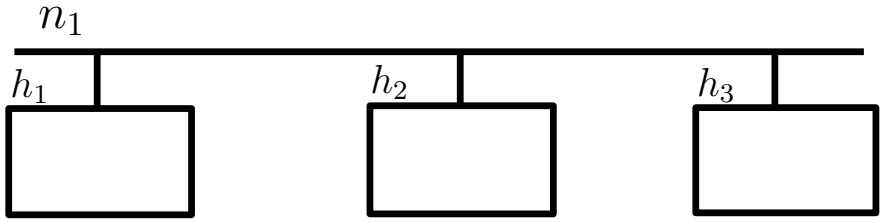

Figure 2 Hardware hosts and network links.

The Component Deployment Problem (CDP) refers to the allocation of software components to the hardware nodes, and the assignment of inter-component communications to network links. Formally, the component deployment problem is defined as $D = \{d \mid d : \mathcal{C} \rightarrow \mathcal{H}\}$, where D is the set of all functions assigning components to hardware resources. One possible assignment of the software components in fig. 1 is shown in fig. 3.

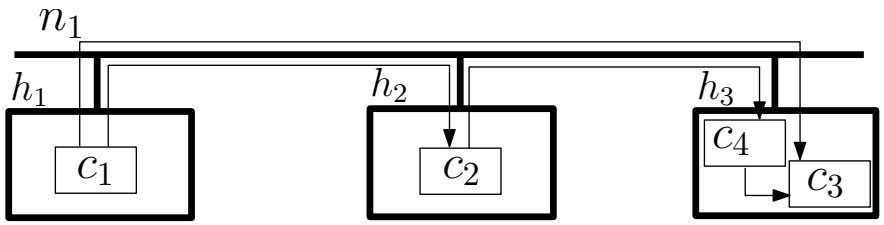

Figure 3 The deployment architecture of software components in fig. 1 and hardware resources in fig. 2.

The way the components are deployed affects many aspects of the final system, such as the processing speed of the software components, how much hardware is used or the reliability of the execution of different functionalities (Meedeniya et al. 2011, Aleti et al. 2009). In this paper, we focus on reliability, since it is one of the most important quality attributes in safety-critical software systems, and at the same time, one of the most complex attributes, expressed with a non-linear function. Reliability can be defined as the probability that the software system produces the correct output Goševa-Popstojanova and Trivedi (2001). In a component deployment system, reliability is a function of the failure rates of software component and hardware architecture.

Two types of failures can occur in a component deployment problem. First, a failure may occur in hardware hosts during the execution of a software component, which affects the reliability of the software modules running on that host. Combined with the fixed and deterministic scheduling strategy, any failure that happens in a host while a software component is executing or queued leads to a system execution failure. The second type of failures happens in data communication channels (network). The communication between two software components over the communication network can cause a failure in the service that depends on this communication. Both failure types are considered into the failure model, and the failure behaviours are used in the computation of reliability.

3.1. Reliability Model for the Component Deployment Problem

The reliability evaluation obtains the mean and variance of the number of executions of components, and combines this information with the failure parameters of the components. Failure rates of *execution elements* can be obtained from the hardware parameters, and the time taken for the execution is defined as a function of the software-component workload and processing speed of its hardware host. The reliability of a component c_i can be computed by Equation 1, where $d(c_i)$ denotes the hardware host where component c_i is deployed.

$$R_i = e^{-\mathrm{fr}_{d(c_i)} \cdot \frac{\mathrm{wl}_i}{\mathrm{ps}_{d(c_i)}}}. \tag{1}$$

where $d(c_i)$ is the deployment function that returns the hardware host where component c_i has been deployed. The reliability of a *communication element* is characterised by the failure rates of the hardware buses and the time taken for communication, defined as a function of the bus data rates dr and data sizes ds required for software communication. The reliability of the communication between component c_i and c_j is defined as

$$R_{ij} = e^{-\mathrm{fr}_{d(c_i)d(c_j)} \cdot \frac{\mathrm{ds}_{ij}}{\mathrm{dr}_{d(c_i)d(c_j)}}}. \tag{2}$$

The probability that a software system produces the correct output depends on the number of times it is executed. For this reason, we calculate the *expected number of executions* for each component $v : C \to \mathbb{R}_{\geq 0}$ as follows:

$$v_i = q_i + \sum_{j \in \mathcal{I}} v_j \cdot p_{ji}, \tag{3}$$

where $\mathcal{I}$ denotes the index set of all components. The transfer probabilities p_{ji} can be written in a matrix form $P_{n \times n}$, where n is the number of components. Similarly, the execution initiation probabilities q_i can be expressed with matrix $Q_{n \times 1}$. The matrix of expected number of executions for all components $V_{n \times 1}$ can be calculated as $V = Q + P^T \cdot V$.

The reliability of a system is also affected by the failure rate of the network. The more frequently the network is used, the higher is the probability of producing an incorrect output. It should be noted that the execution of a software system is never initiated in a network link, and the only predecessor of link l_{ij} is component c_i. Hence, the expected number of executions of network links $v : C \times C \to \mathbb{R}_{\geq 0}$ is calculated as follows:

$$v_{ij} = v_i \cdot p_{ij}. \tag{4}$$

Using the expected number of executions and reliabilities of the communication elements, the reliability of a deployment architecture $d \in D$ is calculated as:

$$R = \prod_{i=1}^{n} R_i^{v_i} \prod_{i,j \text{ (if used)}} R_{ij}^{v_{ij}}. \tag{5}$$

We can convert the reliability function defined in Equation 1 into an integer programming model by introducing decision variables. Initially, we define a binary variable $x_{ij} \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$, such that

$$x_{ij} = \begin{cases} 1 & \text{if the software component } i \text{ is deployed on the hardware unit } j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The decision variable is used to define the failure rate of a hardware unit hosting the software component as follows:

$$\text{fr}_{d(c_i)} = \sum_{j=1}^{m} x_{ij} \text{fr}_j, \tag{7}$$

where fr_j is the failure rate of hardware unit j. For the same purpose, the processing speed of a hardware unit is defined as

$$\text{ps}_{d(c_i)} = \sum_{j=1}^{m} x_{ij} \text{ps}_j, \tag{8}$$

where ps_j is the processing speed of hardware unit j. By substituting these two equations in the original reliability model for a software component defined in Equation 1, we obtain the following integer programming model:

$$R_i = e^{-\sum_{j=1}^{m} x_{ij}\text{fr}_j \frac{\text{wl}_i}{\sum_{j=1}^{m} x_{ij}\text{ps}_j}}. \tag{9}$$

Originally, the reliability of a communication element is characterized by the failure rates of the hardware buses and the time taken for the communication, as defined in Equation 2. Since a software component c_u can be deployed into only one hardware unit, $\sum_{j=1}^{m} x_{uj} = 1$, and $\sum_{j=1}^{m} j x_{uj}$ will result in the index of the hosting hardware unit. As a result, $\sum_{k=1}^{m}\sum_{l=1}^{m} \text{fr}_{kl}\, x_{ik}\, x_{jl}$ will give the failure rate of the network link which connects hardware unit hosting software component c_i with the hardware unit hosting c_j. This is also valid for data rate considerations, i.e. $\sum_{k=1}^{m}\sum_{l=1}^{m} \text{dr}_{kl}\, x_{ik}\, x_{jl}$ estimates the data rate between the hardware hosts that contain components c_i and c_j. As a result, the integer programming equivalent of the reliability for communication elements is equal to

$$R_{ij} = e^{-\sum_{k=1}^{m}\sum_{l=1}^{m} \text{fr}_{kl}\, x_{ik}\, x_{jl} \frac{\text{ds}_{ij}}{\sum_{k=1}^{m}\sum_{l=1}^{m} \text{dr}_{kl}\, x_{ik}\, x_{jl}}}, \tag{10}$$

where ds_{ij} is the communication load from software component i to the software component j.

3.2. Constraints

The problem is naturally constrained, since not all possible deployment architectures can be feasible. For the purpose of this work, we consider four constraints: allocation, colocalisation, memory and communication. These constraints are defined using the decision variable presented in eq. 6.

Allocation constraint takes care of the allocation of all software components into hardware resources. Formally, this constraint is modelled as

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i = 1, \ldots, n. \tag{11}$$

Colocation constraint restricts the allocation of two software components to the same hardware hosts for safety reasons, or forces two components to be deployed into the same hardware unit. For example, in the case when software components c_2 and c_3 are not allowed to be in the same hardware unit, this can be expressed as $x_{2j} + x_{3j} \leq 1 \quad \forall j = 1, \ldots, m$. On the other hand, if they should be deployed into the same hardware unit, the following constraint should be included: $x_{2j} - x_{3j} \leq 0 \quad \forall j = 1, \ldots, m$. For the general case, we define a colocation matrix $C_{n\times n}$, such that $C_{ab} = -1$ means that the software components a and b cannot be in the same hardware unit,

$C_{ab} = 1$ means that the software components a and b should be deployed into the same hardware unit, and $C_{ab} = 0$ is used when we do not care. In essence, the following predicate should be satified at all times:

$$(C_{ab} == -1 \wedge (x_{aj} + x_{bj} <= 1)) \vee (C_{ab} == 1 \wedge (x_{aj} - x_{bj} <= 0)), \forall a, b \in n, \forall j \in m \tag{12}$$

Hardware memory capacity deals with the memory requirements of software components and makes sure that there is available memory in the hardware units. Processing units have limited memory, which enforces a constraint on the possible components that can be deployed into each hardware host. Formally, the memory constraint is defined as

$$\sum_{i=1}^{n} \mathrm{sz}_i x_{ij} \leq \mathrm{cp}_j, \quad \forall j \in \{1, \ldots, m\}. \tag{13}$$

Communication constraint is responsible for the communication between software components. If the transition probability between two software components i and j is positive, $p_{ij} > 0$, these two components will communicate with a certain probability. Therefore, either they should be deployed on the same hardware unit, or on different units that are connected with a communication link (bus) with a positive data rate, $\mathrm{dr}_{ij} > 0$. This is modelled as follows:

$$x_{ik} + x_{jl} \leq 1, \quad \text{if } p_{ij} > 0 \quad \text{and} \quad \mathrm{dr}_{kl} \leq 0. \tag{14}$$

4. A Mixed Integer Linear Programming Model for Reliability Optimisation in the Component Deployment Problem

The reliability function defined is Equation 5 and the non-linear integer programming version in Equation 10 have to be linearised in order to apply linear programming techniques. To do this, we first take the log of both sides of Equation 5, which results in the following equation:

$$\begin{aligned} \log(R) &= \log\Big(\prod_{i=1}^{n} R_i^{v_i}\Big) + \log\Big(\prod_{i,j \text{ (if used)}} R_{ij}^{v_{ij}}\Big) \\ &= \sum_{i=1}^{n} v_i \log(R_i) + \sum_{i,j \text{ (if used)}} v_{ij} \log(R_{ij}). \end{aligned} \tag{15}$$

The application of the concave log function is possible in this case, since the objective function is the product of positive terms. The linearisation changes tge values of the objective function, however, the optimal solution will be the same, despite the change in its objective value.

From Equations 9 and Equation 10, we know that

$$R_i = e^{-\sum_{j=1}^{m} x_{ij} \mathrm{fr}_j \frac{\mathrm{wl}_i}{\sum_{j=1}^{m} x_{ij} \mathrm{ps}_j}}, \quad \text{and} \quad R_{ij} = e^{-\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{fr}_{kl}\, x_{ik}\, x_{jl} \frac{\mathrm{ds}_{ij}}{\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{dr}_{kl}\, x_{ik}\, x_{jl}}},$$

hence, $\log(R_i)$ and $\log(R_{ij})$ are equal to

$$\begin{aligned} \log(R_i) &= -\sum_{j=1}^{m} x_{ij}\mathrm{fr}_j \frac{\mathrm{wl}_i}{\sum_{j=1}^{m} x_{ij}\mathrm{ps}_j}, \\ \log(R_{ij}) &= -\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{fr}_{kl}\, x_{ik}\, x_{jl} \frac{\mathrm{ds}_{ij}}{\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{dr}_{kl}\, x_{ik}\, x_{jl}}. \end{aligned} \tag{16}$$

Substituting $\log(R_i)$ and $\log(R_{ij})$ in eq. 15 with the respective expressions in eq. 16, we get the following expression:

$$\log(R) = \sum_{i=1}^{n} v_i \left[-\sum_{j=1}^{m} x_{ij}\mathrm{fr}_j \frac{\mathrm{wl}_i}{\sum_{j=1}^{m} x_{ij}\mathrm{ps}_j} \right] + \sum_{i,j \text{ (if used)}} v_{ij} \left[-\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{fr}_{kl}\, x_{ik}\, x_{jl} \frac{\mathrm{ds}_{ij}}{\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{dr}_{kl}\, x_{ik}\, x_{jl}} \right].$$

By rearranging the terms that are not indexed by the summation, the objective function can be rewritten in a more compact form as:

$$\log(R) = -\sum_{i=1}^{n} \frac{v_i\, \mathrm{wl}_i \sum_{j=1}^{m} \mathrm{fr}_j\, x_{ij}}{\sum_{j=1}^{m} \mathrm{ps}_j\, x_{ij}} - \sum_{i,j \text{ (if used)}} \frac{v_{ij}\, \mathrm{ds}_{ij} \sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{fr}_{kl}\, x_{ik}\, x_{jl}}{\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{dr}_{kl}\, x_{ik}\, x_{jl}}. \tag{17}$$

The products of the terms in Equation 17 (e.g. $x_{ik}\, x_{jl}$) are the reasons why the function is still non-linear. To convert the reliability function into its equivalent linear programming representation, we perform linearisation transformations based on the product of binary variables and the product of a bounded continuous variable with a binary variable. To linearise the product of the two binary variables x_{ik} and x_{jl}, a new binary variable y_{ijkl} is introduced, defined as:

$$\begin{aligned} &y_{ijkl} = x_{ik} x_{jl} \\ &y_{ijkl} \leq x_{ik},\ y_{ijkl} \leq x_{jl} \\ &y_{ijkl} \geq x_{ik} + x_{jl} - 1 \\ &y_{ijkl} \in \{0, 1\}. \end{aligned} \tag{18}$$

The new constraints introduced above are required to ensure the equivalence between the expressions. With the new variable the objective function is equal to

$$\log(R) = -\sum_{i=1}^{n} \frac{v_i\, w_i \sum_{j=1}^{m} \mathrm{fr}_j\, x_{ij}}{\sum_{j=1}^{m} \mathrm{ps}_j\, x_{ij}} - \sum_{(i,j)} \frac{v_{ij}\, \mathrm{ds}_{ij} \sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{fr}_{kl}\, y_{ijkl}}{\sum_{k=1}^{m}\sum_{l=1}^{m} \mathrm{dr}_{kl}\, y_{ijkl}} \tag{19}$$

To linearise the quotient expressions in Equation 19, two sets of continuous variables u_i and z_{ij} are introduced:

$$\log(R) = -\sum_{i=1}^{n} u_i - \sum_{(i,j)} z_{ij}, \tag{20}$$

where

$$u_i = \frac{v_i\, w_i \sum_{j=1}^{m} \mathrm{fr}_j\, x_{ij}}{\sum_{j=1}^{m} \mathrm{ps}_j\, x_{ij}} \tag{21}$$

$$z_{ij} = \frac{v_{ij}\, \mathrm{ds}_{ij} \sum_{k=1}^{m} \sum_{l=1}^{m} \mathrm{fr}_{kl}\, y_{ijkl}}{\sum_{k=1}^{m} \sum_{l=1}^{m} \mathrm{dr}_{kl},\, y_{ijkl}}. \tag{22}$$

This makes Equation 20 linear. In the next step, we linearise Equation 21, or equivalently,

$$v_i\, w_i \sum_{j=1}^{m} \mathrm{fr}_j\, x_{ij} - \sum_{j=1}^{m} \mathrm{ps}_j\, x_{ij} u_i = 0, \tag{23}$$

where

$$0 \leq u_i \leq \frac{v_i\, \mathrm{wl}_i \max\limits_j \mathrm{fr}_j}{\min\limits_j \mathrm{ps}_j} \quad \text{and} \quad 0 \leq z_{ij} \leq \frac{v_{ij}\, \mathrm{ds}_{ij} \max\limits_{k,l} \mathrm{fr}_{kl}}{\min\limits_{k,l} \mathrm{dr}_{kl}}, \tag{24}$$

where $\max\limits_j \mathrm{fr}_j$ is the maximum failure rate of components, and $\max\limits_{k,l} \mathrm{fr}_{kl}$ is the maximum failure rate of network links. To convert the product of a binary variable into a continuous variable (i.e. $x_{ij}u_i$), we introduce a continuous variable $\bar{u}_{ij} = x_{ij}u_i$. Since the lower bound is equal to 0 and the upper bound is equal to $\frac{v_i\, \mathrm{wl}_i \max\limits_j \mathrm{fr}_j}{\min\limits_j \mathrm{ps}_j}$, the constraints on $\bar{u}_{ij}$ can be written as

$$\bar{u}_{ij} \leq \frac{v_i\, \mathrm{wl}_i \max\limits_j \mathrm{fr}_j}{\min\limits_j \mathrm{ps}_j} x_{ij}, \quad \bar{u}_{ij} \leq u_i, \quad \bar{u}_{ij} \geq u_i - \frac{v_i\, \mathrm{wl}_i \max\limits_j \mathrm{fr}_j}{\min\limits_j \mathrm{ps}_j}(1 - x_{ij}). \tag{25}$$

To linearise Equation 22, we first write it in the following form:

$$v_{ij}\, \mathrm{ds}_{ij} \sum_{k=1}^{m} \sum_{l=1}^{m} \mathrm{fr}_{kl}\, y_{ijkl} - \sum_{k=1}^{m} \sum_{l=1}^{m} \mathrm{dr}_{kl}\, y_{ijkl} z_{ij} = 0. \tag{26}$$

The only non-linear part in Equation 26 is $y_{ijkl}z_{ij}$, which is the product of a binary variable with a bounded continuous variable. Given that $0 \leq z_{ij} \leq \frac{v_{ij}\, \mathrm{ds}_{ij} \max\limits_{k,l} \mathrm{fr}_{kl}}{\min\limits_{k,l} \mathrm{dr}_{kl}}$, we introduce a new set of continuous variables $\bar{z}_{ijkl} = y_{ijkl}z_{ij}$, and three sets of constraints, defined as

$$\bar{z}_{ijkl} \leq \frac{\max\limits_{k,l} \mathrm{fr}_{kl}}{\min\limits_{k,l} \mathrm{dr}_{kl}} y_{ijkl}, \quad \bar{z}_{ijkl} \leq z_{ij}, \quad \bar{z}_{ijkl} \geq z_{ij} - \frac{\max\limits_{k,l} \mathrm{fr}_{kl}}{\min\limits_{k,l} \mathrm{dr}_{kl}}(1 - y_{ijkl}), \quad \forall i, j, k. \tag{27}$$

This final transformation makes the reliability function, a mixed integer linear program. To summarise, the mixed integer linear programming model for reliability optimisation in the component deployment problem is defined as:

$$
MILP \begin{cases}
\max \quad \log(R) = -\sum_{i=1}^{n} u_i - \sum_{i,j \text{ (if used)}}^{m} z_{ij} & \\
\text{such that} & \\
\quad \sum_{j=1}^{m} x_{ij} = 1, \ \ \forall i = 1, \ldots, n & \text{Assignment} \\
\quad x_{aj} + x_{bj} \leq 1, \ \ \forall j = 1, \ldots, m, \ \forall a, b, \ \text{ if } \ c_{ab} = -1 & \text{Colocalization} \\
\quad x_{aj} - x_{bj} = 0, \ \ \forall j = 1, \ldots, m, \ \forall a, b, \ \text{ if } \ c_{ab} = 1 & \text{Colocalization} \\
\quad x_{ik} + x_{jl} \leq 1, \ \text{ if } \ p_{ij} > 0 \ \text{ and } \ \mathrm{dr}_{kl} \leq 0 & \text{Communication} \\
\quad \sum_{i=1}^{n} \mathrm{sz}_i x_{ij} \leq \mathrm{cp}_j, \ \ \forall j \in \{1, \ldots, m\} & \text{Memory} \\
\quad v_i \, \mathrm{wl}_i \sum_{j=1}^{m} \mathrm{fr}_j \, x_{ij} - \sum_{j=1}^{m} \mathrm{ps}_j \, \bar{u}_{ij} = 0 & \\
\quad v_{ij} \, \mathrm{ds}_{ij} \sum_{k=1}^{m} \sum_{l=1}^{m} \mathrm{fr}_{kl} \, y_{ijkl} - \sum_{k=1}^{m} \sum_{l=1}^{m} \mathrm{dr}_{kl} \, \bar{z}_{ijkl} = 0 & \\
\quad y_{ijkl} \leq x_{ik}, \ \ \forall i, j, k, l & \\
\quad y_{ijkl} \leq x_{jl}, \ \ \forall i, j, k, l & \\
\quad y_{ijkl} \geq x_{ik} + x_{jl} - 1, \ \ \forall i, j, k, l & \\
\quad \bar{u}_{ij} \leq \frac{v_i \, \mathrm{wl}_i \max\limits_{j} \mathrm{fr}_j}{\min\limits_{j} \mathrm{ps}_j} x_{ij}, \ \ \forall i, j & \\
\quad \bar{u}_{ij} \leq u_i, \ \ \forall i, j & \\
\quad \bar{u}_{ij} \geq u_i - \frac{v_i \, \mathrm{wl}_i \max\limits_{j} \mathrm{fr}_j}{\min\limits_{j} \mathrm{ps}_j} (1 - x_{ij}), \ \ \forall i, j & \\
\quad \bar{z}_{ijkl} \leq \frac{v_{ij} \, \mathrm{ds}_{ij} \max\limits_{k,l} \mathrm{fr}_{kl}}{\min\limits_{k,l} \mathrm{dr}_{kl}} y_{ijkl}, \ \ \forall i, j, k, l & \\
\quad \bar{z}_{ijkl} \leq z_{ij}, \ \ \forall i, j, k, l & \\
\quad \bar{z}_{ijkl} \geq z_{ij} - \frac{v_{ij} \, \mathrm{ds}_{ij} \max\limits_{k,l} \mathrm{fr}_{kl}}{\min\limits_{k,l} \mathrm{dr}_{kl}} (1 - y_{ijkl}), \ \ \forall i, j, k, l & \\
\quad \bar{z}_{ijkl} \geq 0, \ \ \forall i, j, k, l & \\
\quad y_{ijkl} \in B, \ \ \forall i, j, k, l, \ \text{ where } B \text{ is the Binary set } \{0, 1\} & \\
\quad \bar{u}_{ij} \geq 0, \ \ \forall i, j & \\
\quad u_i \geq 0, \ \ \forall i & \\
\quad z_{ij} \geq 0, \ \ \forall i, j & \\
\quad x_{ij} \in \{0, 1\}, \ \ \forall i, j &
\end{cases}
$$

Linearity is obtained by introducing extra variables and constraints.

5. Experimental Evaluation

The transformation of the nonlinear programming formulation of CDP into MILP introduced more variables and constraints. Hence, we conducted a set of experiments to analyse whether the MILP formulation is competitive with previously known solutions for this problem (Thiruvady et al. 2014). Since the values produced by the new MILP formulation are the log of the ones produced

by the original objective function, the results returned by the MILP solver are converted back to the original non-linear form. This allows a direct comparison with the previous study (Thiruvady et al. 2014).

The MILP was implemented in C++ using CPLEX 12.5 (`www-01.ibm.com/software/commerce/optimization/cplex-optimizer`), which is an efficient tool for solving large mixed integer programs, and widely used in the idustry. Fifteen minutes of execution time was allowed for each instance, which was found to be sufficient to identify optimal solutions when feasibility is found. The experiments were carried out on all the problem instances used by Thiruvady et al. (2014). These problems consist of a number of randomly generated instances with varying complexity and constrainedness. The memory constraint takes its tightness from the ratio of components to hardware hosts - the fewer hosts, the less 'space' there is for components. The smallest instances consist of 15 ECUs and 23 software components whereas the largest instances consist of 60 ECUs and 220 software components. As the emphasis of this work is the ability of the solvers to produce feasible solutions under varying degrees of constraints, the percentages of components with a mutually exclusive colocation constraint was also varied between 10%, 25%, 50%, 75% and 100%. There is no guarantee that feasible solutions exist.

5.1. Benchmark Methods

The results from the MILP solver were compared against ant colony system (ACS) and a hybrid algorithm of constraint programming and ant colony system (CP-ACS). Ant Colony System (ACS) was identified as one of the most successful ACO algorithms (Dorigo and Stűtzle 2004). It uses the pseudo-random proportional rule to determine the next decision in the construction process. In the case of the CDP, a component is chosen randomly and a host is assigned according to Eq. 28, where q is a random number and q_0 a parameter in the range [0,1], while τ_{ij} denotes the pheromone value between component i and ECU j.

$$h_k = \begin{cases} \arg\max\limits_{j \in \mathcal{H}} \tau_{ij}, & \text{if } q \leq q_0, \\ \hat{h}, & \text{otherwise.} \end{cases} \tag{28}$$

$\hat{h}$ uses the distributions determined by the pheromone values $P(h_j) = \tau_{ij} / \sum\limits_{j \in \mathcal{H}} \tau_{ij}$. A solution here is represented by π where the π_i represents the i^{th} software component and its value is a hardware unit, i.e., $\pi_i = j$ is equivalent to $d(i) = j$. The pseudo-random-proportional decision rule is used in the method. A predefined number of n solutions are constructed and added to S. The iteration-best solution is identified and replaces the known best if it is an improvement. If a new solution violates fewer constraints, it replaces the current solution if its fitness is no worse.

Only the component assignments of the best-known solution have their values updated in the pheromone matrix as $\tau_{ij} = \tau_{ij} \cdot \rho + \delta$, where $\delta = \hat{\delta} \times f(s^*)$, where $f(s^*)$ is the fitness of the best solution, and $\hat{\delta}$ is a predefined constant $\delta \in [0.01, 0.1]$. Note that the violations are not considered in the pheromone updates. The parameter ρ is set to 0.1 according to recommendations (Dorigo and Stűtzle 2004) and preliminary testing. Each time an ECU j is selected for component i, a local pheromone update which is typically used with ACS, is applied as $\tau_{ij} = \tau_{ij} \cdot \rho$.

Constraint Programming (CP) models a problem by means of variables and constraints between the variables Marriott and Stuckey (1998). The variables and constraints are maintained by a *constraint solver*. The solver allows enforcing restrictions on the variables and provides feedback on whether these restrictions were consistent with the existing constraints. The feedback is either success (if consistent) or failure (if inconsistent) when a new constraint is added. If successful, the variables' domains are pruned and further constraints may be inferred by filtering algorithms within the CP solver.

As with ACS, software components are incrementally assigned to hardware components. However, when a hardware component is selected, it is tested for feasibility. Here, CP filtering algorithms within the CP solver automatically reduce the domains of current and future components given the past assignments. This amounts to inferring additional constraints which become a part of the set of constraints held in the solver. If successful, the assignment is accepted and discarded otherwise.

The tuning of the parameter values for ACS was performed using a sequential parameter optimisation process (Bartz-Beielstein et al. 2005), which tests each parameter combination using several runs. To decrease the number of tests required, we employ a racing technique, which uses a variable number of runs depending on the performance of the parameter configuration. Parameter configurations are tested against the best configuration so far, using at least the same number of function evaluations as employed for the best configuration.

ACS and CP-ACS are stochastic methods, which may produce different results for the same problem. For this reason, we granted these two algorithms 30 trials, and reported the best solutions and means over the 30 runs. All methods ACS, CP-ACS and MILP were allowed the same amount of time. The implementation, problem instances and results can be downloaded from `users.monash.edu.au/~aldeidaa/MILP`.

5.2. Computational Results

Table 1 shows the results of the MILP model on all the instances (last column). The original ACS and CP-ACS results are shown in the first six columns. The last two columns show the results from the MILP solver and the time needed to solve the model. '-' implies that no feasible solution was found, whereas 'infeasible' means that feasible solutions do not exist.

Table 1 Results for the instances by percentage of interacting components. The original ACS and CP-ACS results are shown in the first six columns. The last two columns show the results from the MILP solver and the time needed to solve the model. '-' implies that no feasible solution was found, whereas 'infeasible' means that feasible solutions do not exist.

	Problem instance	CP-ACS			ACS			MILP	Time
		Best	Mean	Failed	Best	Mean	Failed		
10% Interactions	15 hosts, 23 component	1	0.9999	0%	1	0.9999	3%	0.9998	0.06
	15 hosts, 34 component	0.9999	0.9998	0%	0.9999	0.9999	60%	0.9973	0.05
	33 hosts, 47 component	0.9999	0.9997	0%	1	1	5%	0.9998	0.53
	33 hosts, 51 component	0.9998	0.9996	0%	1	1	0%	0.9996	0.64
	33 hosts, 67 component	0.9994	0.999	5%	1	0.9999	0%	0.9997	1.38
	60 hosts, 120 component	0.9994	0.9989	5%	1	1	0%	0.9992	709.35
	60 hosts, 220 component	-	-	100%	-	-	100%	0.9993	294.6
25% Interactions	15 hosts, 23 components	0.9999	0.9999	0%	1	0.9999	0%	0.9968	0.06
	15 hosts, 34 components	0.9999	0.9998	7%	0.9999	0.9998	0%	0.9996	0.1
	33 hosts, 47 components	0.9992	0.9989	0%	1	0.9999	0%	0.9993	0.95
	33 hosts, 51 components	0.9997	0.9993	0%	1	0.9999	0%	0.9995	1.14
	33 hosts, 67 components	0.9988	0.9983	71%	0.9999	0.9998	0%	0.9983	2.2
	60 hosts, 120 components	-	-	100%	1	0.9999	0%	0.9991	46.79
	60 hosts, 220 components	-	-	100%	-	-	100%	infeasible	898.25
50% Interactions	15 hosts, 23 components	0.9998	0.9998	0%	1	1	0%	0.9999	0.04
	15 hosts, 34 components	0.9999	0.9999	73%	1	1	0%	0.9997	0.14
	33 hosts, 47 components	0.9992	0.9985	2%	1	0.9999	0%	0.9991	1.87
	33 hosts, 51 components	0.9986	0.998	17%	1	0.9999	0%	0.9993	2.21
	33 hosts, 67 components	-	-	100%	1	0.9999	0%	0.9995	15.52
	60 hosts, 120 components	-	-	100%	1	0.9998	0%	0.9989	138.5
	60 hosts, 220 components	-	-	100%	-	-	100%	infeasible	900.7
75% Interactions	15 hosts, 23 component	-	-	100%	0.9999	0.9998	0%	0.9986	0.07
	15 hosts, 34 component	-	-	100%	0.9999	0.9998	23%	0.9986	0.21
	33 hosts, 47 component	-	-	100%	0.9999	0.9999	0%	0.9996	2.26
	33 hosts, 51 component	-	-	100%	0.9999	0.9999	0%	0.9993	3.01
	33 hosts, 67 component	-	-	100%	0.9998	0.9995	0%	0.9983	6.59
	60 hosts, 120 components	-	-	100%	0.9999	0.9997	0%	0.9976	231.92
	60 hosts, 220 components	-	-	100%	-	-	100%	infeasible	901.18
100% Interactions	15 hosts, 23 components	-	-	100%	1	0.9999	0%	0.9987	0.06
	15 hosts, 34 components	-	-	100%	0.9994	0.9991	3%	0.9937	0.23
	33 hosts, 47 components	-	-	100%	0.9999	0.9999	0%	0.9973	2.87
	33 hosts, 51 components	-	-	100%	0.9999	0.9999	0%	0.9972	3.01
	33 hosts, 67 components	-	-	100%	0.9999	0.9998	0%	0.9991	48.28
	60 hosts, 120 components	-	-	100%	0.9999	0.9998	0%	0.9969	248.26
	60 hosts, 220 components	-	-	100%	-	-	100%	infeasible	901.1

As the number of colocation exclusions increases, ACS find feasible solutions even when the hybrid CP-ACS does not. The solutions founds by ACS are usually of better quality than those returned by CP-ACS and MILP solver. This can be explained by the fact that when feasibility is strictly preserved, many attempts at constructing a solution fail when no assignments are possible due to previous conflicting assignments. When infeasible solutions are allowed, like in ACS, the pheromone allocations have a chance of guiding from infeasible to feasible areas incrementally, where solutions of better quality may be found. This advantage only seems to disappear when the search space becomes unmanageably large. Whenever both algorithms find a feasible solution, the quality provided by ACS is at least as good as, and in most cases better. When solution spaces are very constrained, the feasible areas form isolated islands between which the algorithm has difficulty navigating unless it is allowed to traverse the infeasible space. This property of very constrained problems has been observed before (Deb 2001).

Unlike CP-ACS and ACS, the MILP solver finds feasible solutions for all instances where feasible solutions exist. For instance, in one of the larger instances (60 hosts, 220 software components, and 10% interactions), where both CP-ACS and ACS fail at finding any feasible solution, the MILP solver finds a feasible solution of very high quality. The MILP solver takes only 300 seconds to find an optimal solution for this problem instance. In general, the time required for the MILP solver to find solutions is relatively small, which suggests that larger instances may be tackled with the MILP formulation and solved at a reasonable amount of time. However, in the case of the infeasible runs, the entire 15 minutes is used to prove infeasibility. An advantage of the MILP formulation is that it tells us which instances do not have any feasible solutions (c.f. the cells marked as 'infeasible' in Table 1). Regarding solution quality, we find that if solutions are found, ACS is always superior. Hence, we can conclude that the MILP solver is superior to ACS and CP-ACS with respect to dealing with constraints, and is slightly worse with respect to finding solutions of high quality.

6. Conclusions

The reliability optimisation in the component deployment problem that poses itself in software engineering is inherently non-linear. In this study, we presented a mixed integer linear programming model of this problem. Three constraints were considered, one that ensures the memory requirement of a software component is met, a second constraint which restricts the deployment of redundant components to the same host, and a third constraint which enforces that communication between software components is possible. The second constraint leads to 'cul-de-sacs' in the construction of deployments if feasibility is preserved at all costs.

There are two main experimental results relating to feasibility and solution quality. The implementation of the MILP model shows that complex constraints can be solved effectively using

the MILP model, by either finding optimal solutions in relatively short time-frames, or proving infeasibility. The experiments also showed that the quality of the solutions found by solvers in the MILP formulation are usually worse than those found by the ant colony system method.

This is a first step towards modelling the problem presented in this paper as an MILP, hence there is a potential in improving the approximation. As a first improvement, we aim to modify this model to ensure that the optimal objective is achieved. This should help bridge the gap with ACO and improve upon it. Additionally, much larger problems could be considered. In this situation, other heuristics or MILP-based decompositions could prove effective for this problem.

Acknowledgments

The authors gratefully acknowledge the constructive comments and helpful feedback from the anonymous reviewers, which have improved the quality of this work. This research was supported under Australian Research Council's Discovery Projects funding scheme, project number DE 140100017.

References

Afzal, Wasif, Richard Torkar. 2011. On the application of genetic programming for software engineering predictive modeling: A systematic review. *Expert Systems with Applications* **38**(9) 11984–11997.

Aleti, Aldeida. 2015. Designing automotive embedded systems with adaptive genetic algorithms. *Automated Software Engineering* **22**(2) 199–240.

Aleti, Aldeida, Barbora Buhnova, Lars Grunske, Anne Koziolek, Indika Meedeniya. 2013. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering* **39**(5) 658–683.

Aleti, Aldeida, Lars Grunske. 2015. Test data generation with a kalman filter-based adaptive genetic algorithm. *Journal of Systems and Software* **103** 343 – 352.

Aleti, Aldeida, Lars Grunske, Indika Meedeniya, I. Moser. 2009. Let the ants deploy your software - an ACO based deployment optimisation strategy. *International Conference on Automated Software Engineering (ASE'09)*. IEEE Computer Society, 505–509.

Aleti, Aldeida, Indika Meedeniya. 2011. Component deployment optimisation with bayesian learning. *Proceedings of the International ACM Sigsoft Symposium on Component Based Software Engineering*. ACM, 11–20.

Assayad, Ismail, Alain Girault, Hamoudi Kalla. 2004. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. *Dependable Systems and Networks (DSN'04)*. IEEE Computer Society, 347–356.

Bartz-Beielstein, Thomas, Christian Lasarczyk, Mike Preuss. 2005. Sequential parameter optimization. *IEEE Congress on Evolutionary Computation*. IEEE, 773–780.

Bowen, N.S., C.N. Nikolaou, A. Ghafoor. 1992. On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *Computers, IEEE Transactions on* **41**(3) 257–273.

Calinescu, Radu, Marta Kwiatkowska. 2009. Using quantitative analysis to implement autonomic IT systems. *International Conference on Software Engineering, ICSE*. IEEE, 100–110.

Cheung, Roger C. 1980. A user-oriented software reliability model. *IEEE Transactions on Software Engineering* **6**(2) 118–125.

Deb, Kalyanmoy. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

Deb, Kalyanmoy, Amrit Pratap, Sameer Agarwal, T. Meyarivan. 2000. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation* **6** 182–197.

Dimov, Aleksandar, Sasikumar Punnekkat. 2005. On the estimation of software reliability of component-based dependable distributed systems. Ralf Reussner, Johannes Mayer, JudithA. Stafford, Sven Overhage, Steffen Becker, PatrickJ. Schroeder, eds., *Quality of Software Architectures and Software Quality*, *Lecture Notes in Computer Science*, vol. 3712. Springer Berlin Heidelberg, 171–187.

Dorigo, M., T. Stűtzle. 2004. *Ant Colony Optimization*. MIT Press, Cambridge, Massachusetts, USA.

Ernst, Andreas, Houyuan Jiang, Mohan Krishnamoorthy. 2003. Task allocation: A variant of hub location. Tech. rep., CSIRO Mathematical and Information Sciences.

Ernst, Andreas, Houyuan Jiang, Mohan Krishnamoorthy. 2006. Exact solutions to task allocation problems. *Manage. Sci.* **52**(10) 1634–1646.

Feiler, Peter H., Ana-Elena Rugina. 2007. Dependability Modeling with the Architecture Analysis and Design Language (AADL). Tech. rep., CMU/SEI-2007-TN-043.

Fonseca, Carlos M, Peter J Fleming, et al. 1993. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. *ICGA*, vol. 93. 416–423.

Fredriksson, Johan, Kristian Sandström, Mikael Åkerholm. 2005. Optimizing resource usage in component-based real-time systems. *Component-Based Software Engineering, 8th International Symposium*, *LNCS*, vol. 3489. Springer, 49–65.

Gokhale, S.S., T. Philip, P.N. Marinos, K.S. Trivedi. 1996. Unification of finite failure non-homogeneous poisson process models through test coverage. *Software Reliability Engineering. Proceedings., Seventh International Symposium on*. 299–307.

Goševa-Popstojanova, Katerina, Kishor S. Trivedi. 2001. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation* **45**(2-3) 179–204.

Guntsch, Michael, Martin Middendorf. 2003. Solving multi-criteria optimization problems with population-based aco. *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*. Springer, 464–478.

Hadj-Alouane, Atidel Ben, James Bean, Katta Murty. 1999. A hybrid genetic/optimisation algorithm for a task allocation problem. *Journal of Scheduling* **2**(4) 189–201.

Hamlet, Dick, Dave Mason, Denise Woit. 2001. Theory of software reliability based on components. *Proceedings of the 23rd International Conference on Software Engineering*. ICSE '01, IEEE Computer Society, 361–370.

Harman, Mark. 2007a. The current state and future of search based software engineering. Lionel C. Briand, Alexander L. Wolf, eds., *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007*. 342–357.

Harman, Mark. 2007b. Search based software engineering for program comprehension. *15th International Conference on Program Comprehension (ICPC 2007)*. IEEE. Invited paper.

Harris, Inga. 2013. Embedded software for automotive applications. Robert Oshana, Mark Kraeling, eds., *Software Engineering for Embedded Systems*. Newnes, 767 – 816.

Heydarnoori, Abbas, Farhad Mavaddat. 2006. Reliable deployment of component-based applications into distributed environments. *Proceedings of the 3rd International Conference on Information Technology: New Generations*. IEEE Computer Society, IEEE Computer Society.

Kartik, S., C.S.R. Murthy. 1997. Task allocation algorithms for maximizing reliability of distributed computing systems. *Computers, IEEE Transactions on* **46**(6) 719–724.

Krishnamurthy, S., A.P. Mathur. 1997. On the estimation of reliability of a software system using reliabilities of its components. *Software Reliability Engineering, International Symposium on* **0** 146.

Kubat, Peter. 1989. Assessing reliability of modular software. *Operations Research Letters* **8**(1) 35–41.

Kumar, Ranjan, Kazuhiro Izui, Masataka Yoshimura, Shinji Nishiwaki. 2009. Optimal multilevel redundancy allocation in series and series-parallel systems. *Comput. Ind. Eng.* **57**(1) 169–180.

Leen, Gabriel. 2002. Expanding automotive electronic systems. *IEEE Computer* **35** 88–93.

Liang, Yun-Chia, A.E. Smith. 1999. An ant system approach to redundancy allocation. *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 2. –1484 Vol. 2. doi:10.1109/CEC.1999.782658.

Lukasiewycz, Martin, Michael Glaß, Christian Haubelt, Jürgen Teich. 2008. Efficient symbolic multi-objective design space exploration. *ASP-DAC 2008*. IEEE, 691–696.

Mantere, Timo, Jarmo T. Alander. 2005. Evolutionary software engineering, a review. *Applied Soft Computing* **5**(3) 315–331.

Marriott, K., P. Stuckey. 1998. *Programming With Constraints*. MIT Press, Cambridge, Massachusetts, USA.

Martens, Anne, Heiko Koziolek. 2009. Automatic, model-based software performance improvement for component-based software designs. *Formal Engineering approaches to Software Components and Architectures*. Elsevier.

McMinn, Phil. 2004. Search-based software test data generation: a survey. *Softw. Test, Verif. Reliab* **14**(2) 105–156.

Medvidovic, Nenad, Sam Malek. 2007. Software deployment architecture and quality-of-service in pervasive environments. *Workshop on the Engineering of Software Services for Pervasive Environements, ESSPE*. ACM, 47–51.

Meedeniya, I., A. Aleti, I. Avazpour, Ayman Amin. 2012. Robust archeopterix: Architecture optimization of embedded systems under uncertainty. *Software Engineering for Embedded Systems (SEES), 2012 2nd International Workshop on*. 23–29. doi:10.1109/SEES.2012.6225486.

Meedeniya, Indika, Barbora Bühnova, Aldeida Aleti, Lars Grunske. 2011. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software* **84**(5) 835–846.

Mikic-Rakic, Marija, Sam Malek, Nels Beckman, Nenad Medvidovic. 2004. A tailorable environment for assessing the quality of deployment architectures in highly distributed settings. *International Working Conference on Component Deployment (CD'04)*, *LNCS*, vol. 3083. Springer, 1–17.

Moser, I., J. Montgomery. 2011. Population-ACO for the automotive deployment problem. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11, ACM, New York, NY, USA, 777–784.

Moser, I., S. Mostaghim. 2010. The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation. *Evolutionary Computation, IEEE Congress on*. 1–8.

Papadopoulos, Yiannis, Christian Grante. 2005. Evolving car designs using model-based automated safety analysis and optimisation techniques. *Journal of Systems and Software* **76**(1) 77–89.

Reussner, Ralf, Heinz W. Schmidt, Iman Poernomo. 2003. Reliability prediction for component-based software architectures. *Journal of Systems and Software* **66**(3) 241–252.

Sharma, Vibhu Saujanya, Pankaj Jalote, Kishor S. Trivedi. 2005. Evaluating performance attributes of layered software architecture. *Component-Based Software Engineering, 8th International Symposium (CBSE'05)*, *LNCS*, vol. 3489. Springer, 66–81.

Sheikhalishahi, M., V. Ebrahimipour, H. Shiri, H. Zaman, M. Jeihoonian. 2013. A hybrid gapso approach for reliability optimization in redundancy allocation problem. *The International Journal of Advanced Manufacturing Technology* **68**(1-4) 317–338. URL `http://dx.doi.org/10.1007/s00170-013-4730-6`.

Singh, Harshinder, Vittorio Cortellessa, Bojan Cukic, Erdogan Gunel, Vijayanand Bharadwaj. 2001. A bayesian approach to reliability prediction and assessment of component based systems. *International Symposium of Software Reliability Engineering (ISSRE'01)*. IEEE Computer Society, 12–21.

Thiruvady, Dhananjay, I. Moser, Aldeida Aleti, Asef Nazari. 2014. Constraint programming and ant colony system for the component deployment problem. *Procedia Computer Science* **29** 1937 – 1947.

Zhang, Yuanyuan, Anthony Finkelstein, Mark Harman. 2008. Search based requirements optimisation: Existing work and challenges **5025** 88–94.