Constraint Programming and Ant Colony System for the Component Deployment Problem

Dhananjay Thiruvady[1,3], I. Moser[2], Aldeida Aleti[3] and Asef Nazari[1,3]

[1] CSIRO Computational Informatics, Victoria 3169, Australia
dhananjay.thiruvady@csiro.au,asef.nazari@csiro.au
[2] Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia
imoser@swin.edu.au
[3] Clayton School of IT, Monash University, Australia
aldeida.aleti@monash.edu

Abstract

Contemporary motor vehicles have increasing numbers of automated functions to augment the safety and comfort of a car. The automotive industry has to incorporate increasing numbers of processing units in the structure of cars to run the software that provides these functionalities. The software components often need access to sensors or mechanical devices which they are designed to operate. The result is a network of hardware units which can accommodate a limited number of software programs, each of which has to be assigned to a hardware unit. A prime goal of this deployment problem is to find software-to-hardware assignments that maximise the reliability of the system. In doing so, the assignments have to observe a number of constraints to be viable. This includes limited memory of a hardware unit, collocation of software components on the same hardware units, and communication between software components. Since the problem consists of many constraints with a significantly large search space, we investigate an ACO and constraint programming (CP) hybrid for this problem. We find that despite the large number of constraints, ACO on its own is the most effective method providing good solutions by also exploring infeasible regions.

1 Introduction

The automotive industry seeks to enhance the functionalities of vehicles for private transport to improve safety and ease-of-use. Mass-produced Electronic Control Units (ECUs) form the networked hardware infrastructure designed to accommodate the software components that automate and control many different functionalities. Some of these are integral parts of basic vehicle functionality, such as ABS systems, whereas others are designed to enhance user comfort (such as integrated route-guidance systems). New features are considered a luxury at first, but become standard a few years later. Collision warning systems are a recent example of this phenomenon. As the number of features grows, the size of the embedded system that forms the platform for it becomes larger and harder to deploy. Three years ago, the hardware infrastructure of a contemporary car comprised between 50 and 80 ECUs and 3-5 data buses[19]. Typically, the layout of this hardware infrastructure remains the same throughout the lifetime of a car series.

Accommodating the increasing number of software components on this hardware infrastructure is a task of growing complexity. Some components need access to sensors or mechanical devices such as brake mechanisms. Some software components have to have fast communication with other components and therefore have to be deployed to the same ECU or an ECU

that is connected by a fast communication channel. Components which are safety-critical and launched in similar situations cannot be deployed to the same ECU for fear of delaying the response in an emergency.

For a deployment to be feasible, it has to observe these hard constraints. Feasible solutions can be compared using a number of criteria. The reliability of data transmission and communication overhead criteria have been applied in a biobjective formulation [19]. One of the most important conceivable objectives is the reliability of the entire system. In this work, feasible solutions, if found, are compared and reported according to their reliability.

Given the complex nature of the problem, a deterministic approach is unlikely to produce feasible solutions to the problem. Ant Colony Optimisation (ACO) algorithms are constructive optimisers which can preserve feasibility during solution construction. In this work, one of the most successful ACO solvers, Ant Colony System (ACS) is combined with constraint programming (CP) [14] to optimise problem instances with different degrees of constrainedness. CP-ACO has proven effective on problems with non-trivial hard constraints [18, 12, 22, 24, 23]. Adding a local search algorithm to ACO is generally believed to improve the algorithm's performance [7], hence we also explored the alternative of adding a local search to ACS and CP-ACS.

Previously, Population-ACO (P-ACO) [11] was applied to a multiobjective formulation of the automotive deployment problem [20] and found to outperform NSGA-II [6] 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 in this work. To verify whether the hybridisation of ACS with CP brings any benefit, the CP-ACS application is compared with an implementation which uses ACS exclusively.

2 Component Deployment Problem

The architecture of an embedded system represents a model or an abstraction of the real elements of the system and their properties, such as software components, hardware units, interactions of software components, and communications between hardware units.

Formally, we define the software elements as a set of software components, denoted as $\mathcal{C} = \{c_1, c_2, ..., c_n\}$, where $n \in \mathbb{N}$. Each software component is annotated with the following properties:

(a) **Size (sz):** memory size of a component; expressed typically in KB (kilobytes).
(b) **Workload (wl):** computational requirement of a component; expressed in MI (million instructions).
(c) **Initiation probability (q_0):** the probability that the execution of a system starts from the component.

The execution of the software system is initiated in one software component (with a given probability), and during its execution uses many other components connected via communication links, which are assigned with a transition probability [13]. Software interactions are specified for each link from component c_i to c_j with the following properties:

(a) **Data size (ds):** the amount of data transmitted from software component c_i to c_j during a single communication event; expressed in KB (kilobytes).
(b) **Next-step probability (p):** the probability that the execution of component c_i ends with a call to component c_j.

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 capacities of memory, processing power, access to sensors and other peripherals. Each hardware host has the following properties:

(a) **Capacity (cp):** memory capacity of the hardware host, expressed in KB (kilobytes).

(b) **Processing speed (ps):** the instruction-processing capacity of the hardware unit; expressed in MIPS (million instructions per second) [8].
(c) **Failure rate (fr):** characterises the probability of a single hardware unit failure [4] .

The hardware hosts are connected via network links denoted as $\mathcal{N} = \{n_1, n_2, ...n_s\}$, which have the following properties:
(a) **Data rate (dr):** the data transmission rate of the bus; expressed in KBPS (kilobytes per second).
(b) **Failure rate (fr):** failure rate of the exponential distribution characterising the data communication failure of each bus.

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. 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 [17, 1], which constitute the quality attributes of the system. 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.

2.1 Reliability Model for Component Deployment

The reliability evaluation obtains the mean and variance of the number of visits of components in a single execution and combines them 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.

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

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 by Equation 2.

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

Next, the *expected number of visits* of each component $v_c : C \rightarrow \mathbb{R}_{\geq 0}$ is calculated as follows:

$$v_c(c_i) = q_0(c_i) + \sum_{j\in\mathcal{I}}(v_c(c_j)\cdot p(c_j, c_i)) \tag{3}$$

where $\mathcal{I}$ denotes the index set of all components. The transfer probabilities $p(c_i, c_j)$ can be written in a matrix form $P_{n\times n}$. Similarly, the execution initiation probabilities $q_0(c_i)$ can be expressed with matrix $Q_{n\times 1}$. The matrix of expected number of visits for all components $V_{n\times 1}$ can be calculated as $V = Q + P^T \cdot V$.

Next, the expected number of visits of network links $v_l : C \times C \rightarrow \mathbb{R}_{\geq 0}$ is calculated, where $v_l(c_i, c_j)$ denotes the expected number of occurrences of the transition (c_i, c_j). To compute this value, each probabilistic transition $c_i \xrightarrow{p(c_i,c_j)} c_j$ in the model is considered as a tuple of transitions $c_i \xrightarrow{p(c_i,c_j)} l_{ij} \xrightarrow{1} c_j$, the first adopting the original probability and the second having probability $=1$. The expected number of visits of a communication link is computed as:

$$v_l(l_{ij}) = 0 + \sum_{x \in \{i\}} (v_c(c_x) \cdot p(c_x, l_{ij})) \tag{4}$$

Since the execution is never initiated in a link l_{ij} and the only predecessor of link l_{ij} is component c_i, eq. 4 can be reduced to $v_l(l_{ij}) = v_c(c_i) \cdot p(c_i, c_j)$.

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

$$R \approx \prod_{i \in \mathcal{I}} R_i^{v_c(c_i)} \cdot \prod_{i,j \in \mathcal{I}} R_{ij}^{v_l(l_{ij})} \tag{5}$$

2.2 Constraints

In this work, we consider three constraints $\Omega = \{mem, colloc, com\}$, where mem is the memory constraint, loc denotes the localisation constraint and $colloc$ is the collocation constraint.

Memory constraint: Processing units have limited memory, which enforces a constraint on the possible components that can be deployed in each ECU. Formally, let $d^{-1}\colon \mathcal{H} \to \mathcal{C}_h$ denote the inverse relation to $d \in D$, i.e. $d^{-1}(\mathcal{H}) = \{C_h \in C \mid d(C_h) = h\}$. The memory constraint $mem : D \to \{true, false\}$ is defined as:

$$mem(d) \;=\; \forall h \in \mathcal{H} : \sum_{C_h \in d^{-1}(h)} mc(C_h) \;\leq\; mh(h) \tag{6}$$

where $mc(C_h)$ is the total memory required to run the set of components deployed to the hardware host h, and $mh(h)$ is the available memory in host h. Deployment solutions which exceed the available memory in the hardware resources are not allowed.

Colocation constraints: The colocation constraint $coloc : D \to \{true, false\}$ restricts the allocation of two software components to the same hosts for safety reasons.

$$coloc(d) = \forall c \in C : (h \in cr(c_i, c_j) \;\Rightarrow\; d(c_i) \neq d(c_j) \tag{7}$$

where $cr(c_i, c_j)$ is the matrix of colocation restrictions.

Communication constraints: A pair of software components may require communication between each other. These components must be assigned to ECUs which have buses connecting them. This constraint is defined as follows:

$$com(d) = \forall c \in C : (h \in cm(c_i, c_j) \;\Rightarrow\; l_{d(c_i),d(c_j)} \geq 0) \tag{8}$$

where $cm(c_i, c_j)$ is the list of communication restrictions.

3 Previous Work

Papadopoulos and Grante [21] formulated a comprehensive approach to vehicle design which first uses an evolutionary algorithm (EA) to select the functionalities to include in the vehicle model [21], then translates the discovered tradeoffs between profit and cost into software components to be deployed and uses another EA to optimise reliability.

Redundancy allocation, where several instances of the same component are deployed, was studied by Meedeniya, Aleti and Zimmerova [16], who considered responsiveness, reliability and cost of the system in a triobjective formulation optimised using ACO.

Aleti et al. [1] formulated the CDP as a biobjective problem with data transmission reliability and communication overhead as objectives. Memory capacity constraints, location and colocation constraints were considered in the formulation, which was solved using P-ACO [11] as well as MOGA [9]. 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.

A Bayesian learning method was developed by Aleti and Meedeniya [3] and applied to the formulation defined by Aleti et al. [1]. The probabilities of a solution being part of the nondominated set was calculated as the ratio of nondominated solutions produced in the current generation and the overall number of solutions in the generation. Compared to NSGA-II [6] and P-ACO, the Bayesian method was found to produce approximation sets with higher hypervolume values.

Meedeniya et al. [15] applied NSGA-II to the robust optimisation of the CDP considering a varying response time. In reality, vehicles and their ECUs are exposed to temperature differences and similar external factors, which causes the software components to react differently at each invocation. The formulation by Meedeniya et al. treats response time and reliability as probability distributions and presents solutions which are robust with regards to the uncertainty.

Moser and Mostaghim [19] applied NSGA-II to the CDP formulation by Aleti et al. [1] to investigate the performance of alternative constraint handling approaches. Memory and location constraints were considered either using a penalty function, eliminating unfeasible solutions or repairing these. It was observed that repairing unfeasible solutions was by far the most successful method which produced the best solutions across the experiments.

Moser and Montgomery [20] applied P-ACO and NSGA-II to a triobjective variation of the CDP under memory, location and colocation constraints. In addition to the data transmission rate and communication overhead objectives, scheduling time as a measure of system responsiveness was included in the optimisation criteria. Unlike in the biobjective case [1], P-ACO outperformed NSGA-II when combined with a local search which moved a single component at a time if this shift improved the hypervolume contribution of the solution.

4 Methods

4.1 ACS for Component Deployment

Ant Colony System (ACS) was identified as one of the most successful ACO algorithms [7]. 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. 9, 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.

$$h_k = \begin{cases} k = \arg\ max_{j \in \mathcal{H}} \tau_{ij}, & \text{if } q \leq q_0, \\ \hat{h}, & \text{otherwise.} \end{cases} \tag{9}$$

$\hat{h}$ uses the distributions determined by the pheromone values $P(h_j) = \tau_{ij} / \sum_{j \in \mathcal{H}} \tau_{ij}$

Algorithm 1 shows the overview of the approach. 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 ConstructSolution(). A predefined number of n solutions are constructed and added to S. The

Algorithm 1 Ant Colony System

```
INPUT: Component deployment instance 𝒯
while termination conditions not satisfied do
    S = ∅
    for k = 1 to n_ants do
        π_k = ConstructSolution()
        S = S ∪ {π_k}
    end for
    π^ib = argmax{f(π)|π ∈ S}
    π^bs = Update(π^ib)
    𝒯 = PheromoneUpdate(π^bs)
end while
OUTPUT: π^bs
```

Algorithm 2 ConstructSolutionCP()

```
i ← 0, feasible ← true
while i ≤ |𝒞| & feasible do
    i ← i + 1
    D = domain(π_i)
    repeat
        j = selectHC(D,τ)
        feasible = updateJobs(π_i, j, π̂)
        if not(feasible)
        then post(π_i ≠ j)
        D = D \ j
    until D ≠ ∅ ∨ feasible
end while
OUTPUT: π
```

iteration-best solution π^{ib} is identified and replaces the known best π^{bs} (method Update(π^{ib})) if it is an improvement. Update(π^{ib}) compares primarily by constraint violations. 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 π^{bs} have their values updated in the pheromone matrix $\mathcal{T}$ in method PheromoneUpdate(π^{bs}) as $\tau_{ij} = \tau_{ij} \cdot \rho + \delta$, where $\delta = \hat{\delta} \times f(\pi^{bs})$, $\hat{\delta}$ 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 [7] 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 $\tau_{ij} = \tau_{ij} \cdot \rho$. The terminating criterion in this study is a predetermined amount of CPU time - 600 seconds.

4.2 CP-ACS

Constraint Programming (CP) models a problem by means of variables and constraints between the variables [14]. 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.

The CP model used in this study is as follows. For the memory constraint, let $\hat{m}_{ij} = m_i$ if software component i is assigned to hardware component j. Then we impose $\forall j \in \mathcal{H}$: $linear(\hat{m}_{ij}, \leq, M_j)$. This requires that the memory used by all software components assigned to the same hardware components cannot exceed the hardware capacity (M_j). The colocalisation and communication constraints are implemented trivially according to Equations (7) and (8).

In Algorithm 1, ConstructSolution() is replaced by ConstructSolutionCP() (see Algorithm 2). As with ACS, software components are incrementally assigned to hardware components. However, when a hardware component is selected (selectHC($\cdot$)), 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 (line 8).

5 Experiments and Results

5.1 Experimental setup

ACS and CP-ACS were implemented in C++ and the CP component was implemented using GECODE 3.3.1 [10] which has been used in previous CP-ACS implementations (e.g. [24]). All runs were given 600 seconds of CPU time.

5.2 Problem Instances

The instances used for the experiments were created randomly with varying complexity and constrainedness. The memory constraint takes its tightness from the ratio of components to ECUs - the fewer ECUs, the less 'space' there is for components. As the emphasis of this work

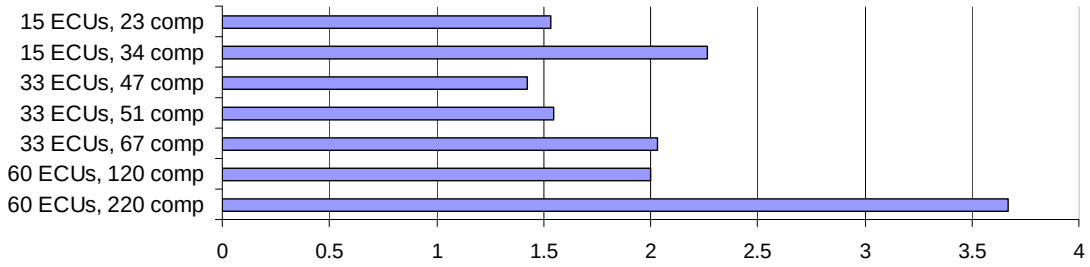

Figure 1: The combinations of ECUs and components used in the experiments. The bars represent the ratios between components and ECUs - the number of components an ECU has to accommodate on average. This indicates the tightness of the memory constraint.

is the ability of the solvers to produce feasible solutions under varying degrees of constraint, 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.

5.3 Results Discussion

Only ACS found solutions to instances when more than 50% of the components were subject to colocation constraints. As Fig. 1 illustrates, the instances with 15 ECUs and 23 components and 33 ECUs and 47/51 components are the least constrained in terms of memory. These instances are solved better by ACS-CP, because it restricts the search space to feasible solutions. Even though these problems are relatively small, ACS seems to be lost among the multitude of possible solutions when there are few colocation constraints to assist in restricting the search space, as the instances with 10% interaction in Table 1 illustrate. The largest instance (60 ECUs and 220 components) is never solved by either algorithm, presumably because of the large search space. It was established in preliminary trials that this instance does have feasible solutions.

As the number of colocation exclusions increases, ACS finds feasible solutions even when the hybrid does not. 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

Table 1: Results for the instances by percentage of interacting components. The best and mean reliability values are given, averaged over 30 runs, as well as the percentage of runs that did not produce a feasible solution.

Algorithm	CP-ACS			ACS		
	Best	Mean	Failed trials	Best	Mean	Failed trials
Colocation constraints between 10% of the components						
15 ECU, 23 Comp	1	0.9999	0%	1	0.9999	3%
15 ECU, 34 Comp	0.9999	0.9998	0%	0.9999	0.9999	60%
33 ECU, 47 Comp	0.9999	0.9997	0%	1	1	5%
33 ECU, 51 Comp	0.9998	0.9996	0%	1	1	0%
33 ECU, 67 Comp	0.9994	0.999	5%	1	0.9999	0%
60 ECU, 120 Comp	0.9994	0.9989	5%	1	1	0%
60 ECU, 220 Comp	0	0	100%	0	0	100%
Colocation constraints between 25% of the components						
15 ECU, 23 Comp	0.9999	0.9999	0%	1	0.9999	0%
15 ECU, 34 Comp	0.9999	0.9998	7%	0.9999	0.9998	0%
33 ECU, 47 Comp	0.9992	0.9989	0%	1	0.9999	0%
33 ECU, 51 Comp	0.9997	0.9993	0%	1	0.9999	0%
33 ECU, 67 Comp	0.9988	0.9983	71%	0.9999	0.9998	0%
60 ECU, 120 Comp	0	0	100%	1	0.9999	0%
60 ECU, 220 Comp	0	0	100%	0	0	100%
Colocation constraints between 50% of the components						
15 ECU, 23 Comp	0.9998	0.9998	0%	1	1	0%
15 ECU, 34 Comp	0.9999	0.9999	73%	1	1	0%
33 ECU, 47 Comp	0.9992	0.9985	2%	1	0.9999	0%
33 ECU, 51 Comp	0.9986	0.998	17%	1	0.9999	0%
33 ECU, 67 Comp	0	0	100%	1	0.9999	0%
60 ECU, 120 Comp	0	0	100%	1	0.9998	0%
60 ECU, 220 Comp	0	0	100%	0	0	100%
Colocation constraints between 75% of the components						
15 ECU, 23 Comp	0	0	100%	0.9999	0.9998	0%
15 ECU, 34 Comp	0	0	100%	0.9999	0.9998	23%
33 ECU, 47 Comp	0	0	100%	0.9999	0.9999	0%
33 ECU, 51 Comp	0	0	100%	0.9999	0.9999	0%
33 ECU, 67 Comp	0	0	100%	0.9998	0.9995	0%
60 ECU, 120 Comp	0	0	100%	0.9999	0.9997	0%
60 ECU, 220 Comp	0	0	0%	0	0	100%
Colocation constraints between 100% of the components						
15 ECU, 23 Comp	0	0	100%	1	0.9999	0%
15 ECU, 34 Comp	0	0	100%	0.9994	0.9991	3%
33 ECU, 47 Comp	0	0	100%	0.9999	0.9999	0%
33 ECU, 51 Comp	0	0	100%	0.9999	0.9999	0%
33 ECU, 67 Comp	0	0	100%	0.9999	0.9998	0%
60 ECU, 120 Comp	0	0	100%	0.9999	0.9998	0%
60 ECU, 220 Comp	0	0	100%	0	0	100%

conflicting assignments. When infeasible solutions are allowed, the pheromone allocations have a chance of guiding from infeasible to feasible solutions incrementally. 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 [5].

ACO algorithms have found to be most successful when combined with a local search method [7]. We repeated the trials with both ACS and CP-ACS with the addition of a hill-climbing approach, still observing the same time limits. The results of CP-ACS improved in one instance but deteriorated in four. The ACS results were improved by the addition of the

local search in two cases but worsened in eleven. This unusual result is best explained by the constrained nature of the problem. In the current ACS formulation, it is unlikely that there exists a continuous path of improvement (fitness improving with every local search move) through feasible space to the local optimum. Infeasible solutions are discarded immediately by the local search, hence the ACS algorithm seems to benefit from constructing solutions rather than attempting to improve existing ones. The same principle applies to the CP-ACS. Since CP-ACS preserves feasibility at all times, it is likely to arrive at a local optimum without the help of a local search. CP-ACS is less successful at finding good solutions overall because the CP module takes a considerable amount of time and therefore covers a smaller portion of the search space.

6 Conclusion

The component deployment problem that poses itself in the automotive industry is extremely constrained. Constraint Programming therefore appears to be a conspicuous approach to finding feasible deployments. In this work, 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.

In a comparison between ACS with and without the use of a CP module, we find evidence of a previous observation [5] - when the search space is extremely constrained, the feasible areas form isolated islands between which the solver finds it hard to navigate, unless it is allowed to cross through an infeasible space.

We believe that this is the reason why ACS by itself outperforms the CP-hybrid especially when the colocation constraint is very tight. Enforcing the constraints strictly prevents initial solutions, which could later be improved, from being created. As the goal is not only to satisfy constraints, but also to optimise the reliability of the solutions, ACS does better in general because it can cross between feasible islands.

In future, we will consider a cross-entropy based method which may also be hybridised with CP, and an IP formulation which can accommodate the large number of constraints effectively.

Acknowledgement

This research was supported under Australian Research Council's Discovery Projects funding scheme, project number DE 140100017.

References

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

[2] Aldeida Aleti, Lars Grunske, Indika Meedeniya, and Irene Moser. Let the ants deploy your software - an aco based deployment optimisation strategy. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 505–509, Washington, DC, USA, 2009. IEEE Computer Society.

[3] Aldeida Aleti and Indika Meedeniya. Component deployment optimisation with bayesian learning. In *Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering*, CBSE '11, pages 11–20, New York, NY, USA, 2011. ACM.

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

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

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

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

[8] Peter H. Feiler and Ana-Elena Rugina. Dependability Modeling with the Architecture Analysis and Design Language (AADL). Technical report, CMU/SEI-2007-TN-043, 2007.

[9] Carlos M Fonseca, Peter J Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *ICGA*, volume 93, pages 416–423, 1993.

[10] Gecode. Gecode: Generic Constraint Development Environment. Available from http://www.gecode.org, 2010.

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

[12] M. Khichane, P. Albert, and C. Solnon. CP with ACO. *Lecture Notes in Computer Science*, 5015:328–332, 2008.

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

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

[15] I. Meedeniya, A. Aleti, I. Avazpour, and Ayman Amin. Robust archeopterix: Architecture optimization of embedded systems under uncertainty. In *Software Engineering for Embedded Systems (SEES), 2012 2nd International Workshop on*, pages 23–29, 2012.

[16] Indika Meedeniya, Aldeida Aleti, and Barbora Buhnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium of Avionics/Automotive Systems Engineering (SAASE09), San Diego, CA*, 2009.

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

[18] B. Meyer and A. Ernst. Integrating ACO and Constraint Propagation. *Lecture Notes in Computer Science: Ant Colony, Optimization and Swarm Intelligence*, 3172/2004:166–177, 2004.

[19] I. Moser and S. Mostaghim. The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, 2010.

[20] Irene Moser and James Montgomery. Population-aco for the automotive deployment problem. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 777–784, New York, NY, USA, 2011. ACM.

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

[22] D. Thiruvady, C. Blum, B. Meyer, and A. T. Ernst. Hybridizing Beam-ACO with Constraint Programming for Single Machine Job Scheduling. *Lecture Notes in Computer Science*, 5818:30–44, 2009.

[23] D. Thiruvady, G. Singh, A. T. Ernst, and B. Meyer. Constraint-based ACO for a Shared Resource Constrained Scheduling Problem. *International Journal of Production Economics*, 141(1):230–242, 2012.

[24] D. R. Thiruvady, B. Meyer, and A. T. Ernst. Car Sequencing with Constraint-based ACO. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 163–170, New York, USA, 2011. ACM.