Population-ACO for the Automotive Deployment Problem

Irene Moser
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Vic, Australia
imoser@swin.edu.au

James Montgomery
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Vic, Australia
jmontgomery@swin.edu.au

ABSTRACT

The automotive deployment problem is a real-world constrained multiobjective assignment problem in which software components must be allocated to processing units distributed around a car's chassis. Prior work has shown that evolutionary algorithms such as NSGA-II can produce good quality solutions to this problem. This paper presents a population-based ant colony optimisation (PACO) approach that uses a single pheromone memory structure and a range of local search operators. The PACO and prior NSGA-II are compared on two realistic problem instances. Results indicate that the PACO is generally competitive with NSGA-II and performs more effectively as problem complexity—size and number of objectives—is increased.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms, Performance

Keywords

Ant colony optimisation, automotive deployment, constrained problem, embedded systems, genetic algorithms, local search, multiobjective problem, optimisation

1. INTRODUCTION

The hardware infrastructure of a contemporary car is an embedded system with buses connecting groups of processing units which can be used to host software components performing related tasks. Passenger vehicles tend to be equipped with system and multimedia buses which connect hardware hosts for distributed software components which collectively perform tasks pertaining to these functions. A bus can have up to 60 attached processing units.

Car manufacturers continue to develop new value-adding features for integration into cars, such as parking assistance and traffic guidance systems. Consequently, the number of software components to accommodate continues to grow, while the embedded infrastructure of any given car model remains the same until a new series is launched. Generally, depending on the processing and memory capacities, hardware units can accommodate up to four software components. Usually, however, 1–2 component assignments are the norm.

The automotive deployment problem (ADP) is that of assigning software components to hardware units such that the solution obtained is the best possible option to implement. Several objectives ought to be optimised for an ideal deployment. For example, data communication should be reliable, which puts demands on the proximity of components which have a need to communicate. Also, components can communicate most reliably when they are located on the same hardware unit, but waiting times may be incurred when too many components are allocated to the same processing unit.

The ADP has previously been optimised using an implementation of NSGA-II [13, 18]. This work adapts the population-based niching Ant Colony Optimisation (PACO) devised by Guntsch and Middendorf [9] to the ADP. Various local search moves are combined with this technique and the experimental evaluation compares the outcomes with the help of the hypervolume indicator as a quality measure.

2. SOFTWARE DEPLOYMENT IN AN AUTOMOTIVE SETTING

The automotive deployment problem described here represents one aspect of the optimisation task of allocating the software components that provide the control logic for contemporary vehicles (e.g., ABS and airbag control functions) to processing units distributed around the vehicle. The necessary software functionality is represented by a predefined set of software components, which have to be deployed to the hardware units. A solution to the automotive deployment problem is a mapping of all available software components C to all or a subset of hardware units U: $d_i = \{(c_1, u_{i_1}), (c_2, u_{i_2}), \ldots, (c_n, u_{i_m})\}$, where i_1 to i_n are integers in $[1, m]$. The set of all possible solutions is $D = \{d \mid d : C \rightarrow U\}$. For ease of notation, the set of indices of components assigned to hardware units u_j is denoted C_{u_j}. More in-depth description of the problem and its aspects is found in [2, 7, 8].

2.1 Hardware Infrastructure

Hardware units are connected to different data buses and vary in memory capacity, processing power and failure propensity. Data buses are characterised by different data rates and degrees of reliability. The speed and reliability of communication between two hardware units therefore depends on the data buses to which they are connected. More formally, the hardware architecture is defined in the following terms:

- The set of available hardware units $U = \{u_1, u_2, ..., u_m\}$, $m \in \mathbb{N}$;
- capacity of a unit, $cp : U \to \mathbb{N}$;
- processing speed, $ps : U \to \mathbb{N}$, being how many million machine instructions are processed per second;
- failure rate, $fr : U \to \mathbb{R}$;
- data rate of the preferred bus, $dr : U \times U \to \mathbb{N}$;
- network delay, $nd : U \times U \to \mathbb{R}$;
- reliability of the preferred bus, $rel : U \times U \to \mathbb{R}$.

2.2 Software

The optimisation task is to deploy a number of predefined software components to the hardware units in the existing hardware infrastructure. Each software component fulfils a predefined function in the vehicle. Components have the following properties:

- The set of components, $C = \{c_1, c_2, ..., c_n\}$, $n \in \mathbb{N}$;
- component size in memory, $sz : C \to \mathbb{N}$;
- estimated execution length, expressed as a fraction of one million machine instructions, $mi \in \mathbb{R}$;
- location restriction, $lr : C \to \mathcal{P}(U)$;
- colocation restriction, $coloc : C \times C \to 1, -1$;
- data size sent over a link, $ds : C \times C \to \mathbb{R}$;
- frequency of communication between two components, $freq : C \times C \to \mathbb{R}$;
- the communication link between two components i and j, $l_{ij} = (c_i, c_j)$.

In reality, a component may spend different portions of its processing time supporting various services. However, in the present work this has been simplified to the single execution length, mi.

2.3 Objective Functions

As this is a practical problem faced by car manufacturers and their engineers, the conceivable set of objectives is large. The availability of objectives depends on investigative work undertaken by researchers in the field of ADP [2] who interview engineers and formalise the objectives.

Data Transmission Reliability (DTR) follows the definition of Malek [11]. Reliability of the data transmission is a crucial quality attribute in a real-time embedded system, where important decisions are taken based on the data transmitted through the communication links. The Data Transmission Reliability (DTR) formulation we use has first been defined by Malek [11].

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

In embedded systems, with their constrained hardware resources, repeated transmissions between software components are discouraged. The Communication Overhead (CO) objective [12] attempts to enforce minimal data communication for a given set of components and system parameters. As a network- and deployment-dependent metric, the overall communication overhead of the system is used to quantify this aspect. It was first formalised by Medvidovic and Malek [12].

$$f_{CO}(d) = \sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j) \cdot nd(d(c_i), d(c_j)) + \\ + \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{freq(c_i, c_j) \cdot ds(c_i, c_j)}{dr(d(c_i), d(c_j)) \cdot rel(d(c_i), d(c_j))} \qquad (2)$$

Component responsiveness is an important aspect of an embedded system, and is affected by the number and nature of other components with which hardware processing time must be shared. This aspect of a deployment is measured by taking the average processing time required for a hardware unit to execute all of its allocated components in a round-robin fashion. This minimisation objective, denoted Scheduling Time (ST), is given by

$$f_{ST}(d) = \frac{1}{m} \cdot \sum_{j=1}^{m} \left(\frac{\sum_{i \in C_{u_j}} mi_i}{ps_j} \right). \qquad (3)$$

2.4 Constraints

The ADP is subject to several hard constraints which, when violated, render a system solution worthless as an alternative for implementation. They are therefore considered separately from the objectives and enforced during the optimisation.

The number of components to be placed on a single hardware unit is restricted by the memory size of the unit and the memory requirements of the component. In analogy with the traditional bin packing problem, the memory constraint Ω_{mem} is defined as follows:

$$\Omega_{mem}(d) \ = \ \forall u \in U : \sum_{i \in C_{u_j}} sz(c_i) \ \leq \ cp(u_j) \qquad (4)$$

The location constraint Ω_{loc} allows only a subset of the available hardware units to be considered as allocation alternatives. More recent exchanges with industry partners lead us to believe that at least for some car manufacturers, this constraint reflects the necessity of confining groups of interacting components to hardware on the same bus.[1] Depending on the manufacturer, cars may have a bus reserved for multimedia application components and a bus for the control functions of the vehicle.

$$\Omega_{loc}(d) = \forall c \in C : u \in lr(c) \Rightarrow d(c) \neq u \qquad (5)$$

[1]M. Scott, personal communication, 8 July 2010

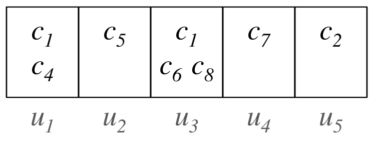

Figure 1: Data structure used by all algorithms

The colocation constraint Ω_{coloc} excludes certain components from residing on particular hardware units. Components responsible for mission-critical functions must not be at risk of failing at the same time.

$$\Omega_{coloc}(d) = \forall c \in C : c_i, c_j \in coloc_{-1} \Rightarrow d(c_i) \neq d(c_j) \quad (6)$$

3. ALGORITHMS

3.1 Data Structure

All algorithms use a data structure that represents the assignment of one or more components to a hardware unit. The data structure is illustrated in Figure 1, a mapping from hardware units to a set of assigned components.

3.2 NSGA-II

The improved form NSGA-II of the Nondominated Sorting Genetic Algorithm (NSGA) by Srinivas and Deb [4] is considered one of the most successful state-of-the-art multiobjective problem solvers. The NSGA uses the mutation and recombination mechanisms typical of GA implementations in combination with a nondominated sorting mechanism which divides the existing population into successive approximation sets. The sets are ranked from best to worst. The ranking is subsequently used as a means of comparison for individuals when performing tournament selection for the mating pool.

The NSGA-II used for comparison here has previously been used to solve the ADP [13, 18] and has thus been customised for this particular constrained assignment problem. Three genetic operators are used: a point mutation operator, which reassigns a single component to another hardware unit; a swap mutation operator, which swaps two component/host assignments subject to compatibility; and a customised crossover operator. The recombination (i.e., crossover) procedure initially combines two individuals by combining the lists of components for each hardware unit. At this stage, each hardware unit has, on average, double the assignments it would usually have. Duplicate assignments are possible, because two identical sets of components, originating from two solutions, are assigned to the hardware units. In the next step, two individuals are formed again by dividing the hardware-specific lists into two component lists to be allocated to the same hardware unit in offspring 1 and offspring 2 respectively. Further details are available in the previous works [13, 18].

3.3 Population-ACO

Ant Colony Optimisation (ACO) is a constructive search algorithm which, similarly to Population-Based Incremental Learning (PBIL), uses probabilities to choose among options during the constructive steps. The probabilities are encoded as pheromone values. In the case of an assignment problem, it is most effective to assign the probability values to a component–hardware unit pair [15, 16]. Thus, when a new deployment is constructed component by component, the 'pheromone' values bias the choice of hardware unit for the respective component. Conceivably the most successful adaptation of the ACO paradigm to multiobjective optimisation is Population-ACO (PACO) devised by Guntsch and Middendorf [9]. Unlike typical single-objective ACO algorithms, the pheromone values are created anew at each iteration based on selected members of the population. PACO uses an implicit niching mechanism in that it chooses the k nearest neighbours of a randomly selected individual as a basis for the pheromone values corresponding to the component–hardware unit associations used in those individuals. In our implementation, the neighbourhood is defined in objective space using an ordering by a randomly chosen objective.

Unlike Guntsch and Middendorf [9], the PACO algorithm described here uses a single set of pheromone values instead of one for each objective. It also uses a fixed pheromone addition value, which removes the effect of weightings.

The PACO algorithm in this work incorporates a relatively greedy choice of next assignment, the pseudo-random proportional rule [5], which chooses the assignment with the maximal pheromone value with a probability of q_0. With probability $1 - q_0$ the assignment is chosen randomly in proportion to the pheromone value. This mechanism is most effective when good solutions are located in the immediate vicinity of an existing solution.

3.4 Stochastic Local Search

ACO researchers generally agree that the ACO paradigm will perform competitively only in combination with a local search (LS) [6]. If the problem is of limited complexity with a fitness landscape comprising long gradients, the local search itself may prove very effective as a method in its own right. Hence the purpose of implementing a local search for this work is twofold: by itself, the local search or hill climber can be seen as a baseline comparison; in combination with PACO it adds a local search component that complements the global search performed by ACO.

The local search move is similar to mutation in that it moves a uniformly randomly chosen component to a different, uniformly randomly chosen hardware unit. In a single-objective environment, the definition of 'better' is usually understood as surpassing the fitness of the original solution to which the local search step was applied. In a multiobjective environment, this acceptance criterion is too restrictive—the more objectives exist, the less likely it is for the new individual to dominate the original.

Basseur and Burke [3] built on the work of Zitzler et al. [21] and compared the use of different binary indicators which can quantify the improvement of one solution over another, or indeed of a set of solutions over another set. Thus, their indicator-based multiobjective local search is capable of establishing whether a solution is 'good enough' to be added to a set of existing solutions.

As in the case of the ADP, there is no necessity for a crowding mechanism and all solutions that are not dominated by the current approximation set may be added to it. Hence, the local search acceptance criterion is whether the newly created solution qualifies as part of the nondominated set.

Three local search moves were explored in the course of these experiments. The simple move is analogous to the mutation operator used by the NSGA-II: a simple move reassigns one single component to a different hardware unit. NSGA-II's swap move proved less useful for hill climbing, as it dislodges good assignments when the overall fitness of the solution is already high. Therefore, only the simple move was used for hill climbing. In addition, a 'flexible' local search move (referred to as LS shift) was implemented for this purpose. The shift move chooses a random component to relocate, then picks a random hardware unit to reassign the component to. If this hardware unit has too many assignments (with respect to the memory capacity) already, or is already hosting incompatible components, these are shifted to other hardware units.

4. EXPERIMENTS

4.1 Trial Parameters

Each trial was repeated 30 times to ensure statistical significance. Regardless of algorithm, each of the 30 trials was allowed an equal number of 100 000 function evaluations. The results were recorded as the objective function values for final approximation set for each trial.

For the final approximation set, the hypervolume metric, recommended by Zitzler et al. [22], was reported. The development of the hypervolume of the approximation set was also recorded for every 500^{th} function evaluation. These values were collected throughout the 30 trials and averaged.

4.2 Problem Instances

Two problem instances were used in these experiments. Both have an infrastructure of 60 hardware units on a single bus. According to one manufacturer, software components do not cooperate across buses, therefore the component allocations along each bus can be treated as an individual problem.[2] In contemporary passenger vehicles, we cannot expect to encounter infrastructures with more than 60 hardware units residing on a single bus. Therefore, the difficulty of the instances is sufficient to tax the capability of the solver to the full satisfaction of contemporary car manufacturers.

One of the problem instances (H60C120) used has 120 software components which are to be deployed on the 60 hardware units, the other (H60C220) has 220. Due to their memory capacities, the hardware units can accommodate four components on average. Thus the larger problem is close to the limits of feasibility.

4.3 Algorithmic Adaptations

4.3.1 NSGA-II

The parameter values in Table 1 were used for the trials with NSGA-II. Previous experience has shown that the approximation set, the first front found by the nondominated sorting algorithm, never exceeds a population size of 50. Hence there is no danger of discarding valuable information and no need for a crowding mechanism as implemented in the original NSGA-II [4]. The mating pool size of 100% of the population is selected for reproduction using binary tournament ($t = 2$). Reproduction creates 45 new individuals ($g = 0.9$, i.e. 90% of the population size.) Crossover is

[2]M. Scott, personal communication, 8 July 2010

Table 1: Parameters used for NSGA-II

Attribute	Value	Explanation
n	50	population size
m	1	mating pool ratio
g	0.9	regeneration rate
r_c	1	crossover rate
r_m	0.5	mutation rate
q_0	0.9	Proportion of greedy choices
t	2	tournament size

Table 2: Parameters used for PACO

Attribute	Value	Explanation
n	50	solutions per cycle
k	3	solutions for pheromone update
τ_0	0.01	initial pheromone value
Δ	$\frac{0.9}{k}$	pheromone addition
q_0	0.9	Proportion of greedy choices

always performed and in 50% of the cases, both point and swap mutations ensue.

4.3.2 PACO

The parameter settings in Table 2 produced the best outcomes in preliminary trials. At every iteration, the algorithm produces 50 solutions. The process starts with the pheromone update. All possible assignment combinations are set to τ_0. The existing approximation set is ordered by one randomly chosen objective. One random individual is chosen as well as its $k-1$ nearest neighbours in the sorted list. The amount Δ of pheromone is added to a component/hardware unit assignment for each occurrence in one of the k solutions. Then, n solutions are constructed by traversing a list of components and assigning them to hardware units. With probability q_0, the assignment with the highest pheromone value is chosen. The choice is made randomly with the given pheromone distribution with probability $1 - q_0$. The newly constructed individuals which are not dominated by the existing approximation set are added to the set. If local search is used, these solutions are then submitted to the local search phase.

The LS used by PACO employs the same changes as the Hill Climber (HC), but it only applies a single LS step to each solution in the nondominated set once in each iteration after the ACO algorithm has created n new solutions according to the pheromone distribution.

In constrained assignment problems the order in which items—*components* in this instance—are assigned can affect the likelihood of completing a feasible solution, as previous assignments can leave no viable options for later items [14, 19]. A common high-level heuristic is to assign constrained items early. Thus, initial tests were conducted with a several different assignment orders: the default order in which software components are described in the problem; a dynamically randomised order for each new solution; and by ordering components by non-increasing memory requirement. These tests revealed that, while the ADP is constrained, the ACO algorithm was generally able to produce feasible solutions without the assistance of the heuristic assignment order, and indeed was able to produce improved results using the dynamically randomised order. This sug-

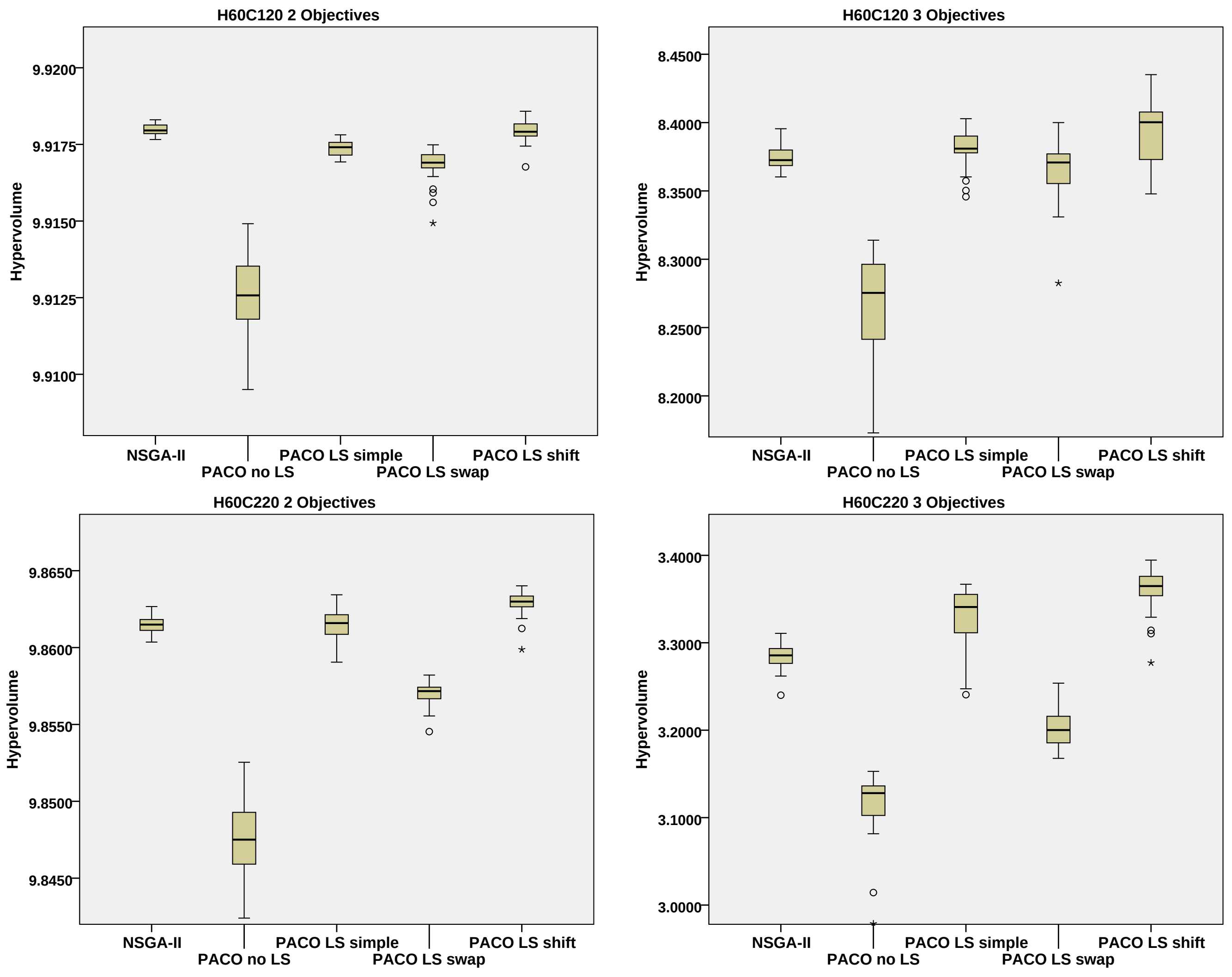

Figure 2: Hypervolumes of the result sets of all algorithms from the four experiments. The mean, interquartile range and minimum/maximum values are shown. Disconnected items are outliers.

gests that, if heuristically determined assignment orders are to be used with this problem, then the heuristic should attempt to encourage good placement of related components rather than attempting to encourage merely feasible solutions. Such assignment orders will be investigated in future work. All PACO results reported here used the dynamically randomised assignment order.

4.3.3 Hill Climbing

The stochastic local search algorithm is used in combination with a solution generator which constructs initial solutions by assigning components to randomly selected hardware units. The assignments are made subject to the problem's constraints, and the hill climbing algorithm receives feasible solutions. Unlike the LS combined with PACO, the HC attempts to improve each solution until no further improvement is possible.

The only parameter required by the local search is the 'stop criterion', which determines when a solution can no longer be improved. The stop criterion was set to 30 consecutive failed attempts. Whenever an improvement is found, the counting for the stop criterion restarts.

5. RESULTS

The results shown in Figure 2 describe the result sets of trials with NSGA-II, the PACO algorithm without a local search, the PACO algorithm with 'simple' LS, with the swapping LS and with the flexible LS. The results of the HC trials were not included in the box plots. The mean qualities achieved by the HC are significantly lower than even those of the PACO solver when no LS is used. Table 3 also shows that the shift move lends itself to hill climbing and produces even better results than the simple move.

Given these results, we conclude that the problem is too complex to be solved well by a stochastic hill climber. We assume the fitness function is linear only in small sections at a time, which offers no gradient for hill climbing moves. Implementing stochastic instead of deterministic LS, i.e. randomly choosing components to move and accepting the first improvement instead of trialling all moves and choosing the

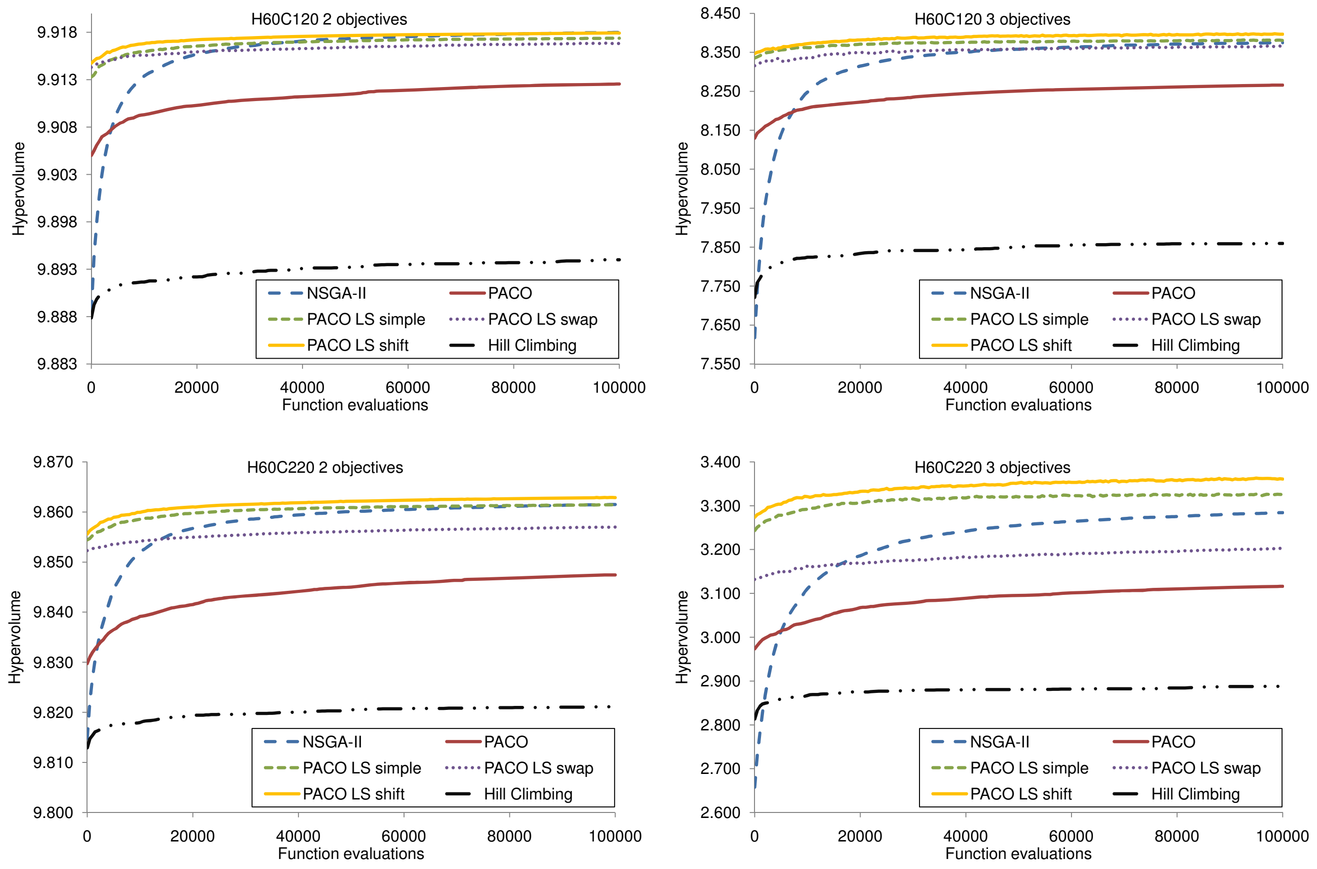

Figure 3: Hypervolume developments.

Table 3: Mean Results for HC Approaches, Compared to PACO without LS

	HC simple	HC shift	PACO, no LS
H60C120/2	9.8939	9.8937	9.9125
H60C120/3	7.8595	7.8616	8.2657
H60C220/2	9.8211	9.8206	9.8474
H60C220/3	2.8878	2.8909	3.1162

largest improvement, could not alleviate this problem. Too many moves (>85%) are unsuccessful according to our observations. Consequently, we dismissed the HC as a viable optimiser and did not subject it to statistical testing.

The most successful approaches in all four experimental settings are NSGA-II, PACO with the flexible LS move and PACO with the simple LS move. The swap move performs consistently worse. This becomes more evident with growing numbers of function evaluations and growing numbers of components in the problem. When the overall solution quality is very high, a successful swap move seems to dislodge advantageous assignments at the same time as it finds new favourable assignments. Hence, shifting two components at a time seems counterproductive.

Interestingly, the flexible LS performs slightly better than even the simple LS move when combined with PACO. The difference between the swap and flexible moves is that the flexible move will only move a component out of a relocating component's way if constraints prevent both from residing on the same hardware.

Another interesting discovery is the fact that when combined with flexible or simple LS, PACO can outperform NSGA-II, which is considered one of the most successful solvers for multi-objective problems. The difference in performance grows significantly with the number of objectives, even more so than with the complexity of the problem. However, as assumed in the literature, PACO only produces competitive results when paired with a LS.

ACO implementations have been observed to outperform Genetic Algorithms (GA) before in single-objective [10] and multiobjective environments [1, 17, 20]. We can confirm these findings.

The hypervolume development in Figure 3 pertaining to the same trials shows that within the first 20 000 function evaluations, NSGA-II produces poor results compared to all LS-enhanced PACO approaches. This is a clear disadvantage when function evaluations are expensive.

The Kolmogorov-Smirnov test for normality confirms that not all result sets are normally distributed. Some algorithms' result sets have a normal distribution, but not for all experiments. Hence, we applied the nonparametric Mann-Whitney U test for pairwise comparisons of the NSGA and the two most successful PACO implementations, one of which uses simple LS, the other flexible LS. For the three best-performing approaches, we would like to establish whether the groups are significantly different. The pairwise compar-

Table 4: Mann-Whitney U Test

	NSGA / PACO,simple	NSGA / PACO, shift	PACO,simple / PACO,shift
H60C120/2	.00	.53	.00
H60C120/3	.01	.00	.00
H60C220/2	.81	.00	.00
H60C220/3	.00	.00	.00

isons are shown in Table 4. There is a significant difference between all pairs except NSGA-II and PACO using flexible LS when solving the smaller problem with two objectives and NSGA-II and PACO with simple LS applied to the larger two-objective problem. All other combinations perform differently on a significance level <1%. Regarding the comparison between PACO using flexible move and simple move, the significant difference is always in favour of the approach with the flexible move. When NSGA-II is compared to the PACO implementations, NSGA-II only wins when fewer objectives are optimised, preferably on a smaller problem. The box plots illustrate the 'draw' represented by the significance level of 0.81 (not significant); the problem is larger but there are only two objectives to optimise, which puts NSGA-II on a par with PACO and simple LS.

6. CONCLUSIONS

The Population-ACO approach with its inherent niching mechanism outperforms the state-of-the art NSGA-II when combined with a suitable LS that makes small moves. The difference in performance increases with the number of objectives and the size of the problem. The PACO approaches with LS provide vastly superior quality at a very early stage of the search. However, the development of the approximation sets' hypervolumes seems to suggest that even though solution quality improvement is possible even after 200 000 function evaluations, the solvers seem to improve their solution fitness in the same proportion.

For future work we also plan to investigate the correlation between the number of objectives and the relative performance of the ACO algorithm compared to other methods.

7. REFERENCES

[1] I. Alaya, C. Solnon, and K. Ghedira. Ant colony optimization for multi-objective optimization problems. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 450–457, 2007.

[2] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In *Model-based Methodologies for Pervasive and Embedded Software (MOMPES)*, pages 61–71, 2009.

[3] M. Basseur and E. Burke. Indicator-based multi-objective local search. In *2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 3100–3107, 2007.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2002.

[5] M. Dorigo and L. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[6] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[7] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In *International Conference on Software Engineering, ICSE*, pages 849–852, 2006.

[8] L. Grunske. Early quality prediction of component-based systems — A generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[9] M. Guntsch and M. Middendorf. Solving multi-criteria optimization problems with population-based ACO. In C. Fonseca, P. Fleming, E. Zitzler, L. Thiele, and K. Deb, editors, *Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science*, pages 464–478. Springer, 2003.

[10] F. Luna, C. Blum, E. Alba, and A. J. Nebro. ACO vs EAs for solving a real-world frequency assignment problem in gsm networks. In *Proceedings of the 9th Annual Conference on Genetic and evolutionary computation*, GECCO '07, pages 94–101, New York, NY, USA, 2007. ACM.

[11] S. Malek. *A User-Centric Approach For Improving A Distributed Software System's Deployment Architecture*. PhD dissertation, University of Southern California, 2007.

[12] N. Medvidovic and S. Malek. Software deployment architecture and quality-of-service in pervasive environments. In *Workshop on the Engineering of Software Services for Pervasive Environments, ESSPE*, pages 47–51. ACM, 2007.

[13] J. Montgomery and I. Moser. Parallel constraint handling in a multiobjective evolutionary algorithm for the automotive deployment problem. In *2010 IEEE International Conference on e-Science Workshops*, pages 104–109, Brisbane, Australia, 2010.

[14] J. Montgomery, M. Randall, and T. Hendtlass. Search bias in constructive metaheuristics and implications for ant colony optimisation. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *4th International Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2004*, volume 3172 of *Lecture Notes in Computer Science*, pages 390–397, Brussels, Belgium, 2004. Springer-Verlag.

[15] J. Montgomery, M. Randall, and T. Hendtlass. Automated selection of appropriate pheromone representations in ant colony optimisation. *Artificial Life*, 11(3):269–291, 2005.

[16] J. Montgomery, M. Randall, and T. Hendtlass. Solution bias in ant colony optimisation: Lessons for selecting pheromone models. *Computers & Operations Research*, 35(9):2728–2749, 2008.

[17] N. Mortazavi, G. Kuczera, and L. Cui. Comparison of genetic algorithm and ant colony optimization methods for optimization of short-term drought mitigation strategies. In *Hydroinformatics in hydrology, hydrogeology and water resources*. Springer, 2009.

[18] I. Moser and S. Mostaghim. The automotive deployment problem: A practical application for

constrained multiobjective evolutionary optimisation. In *2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 4272–4279. IEEE, 2010.

[19] M. Randall. Heuristics for ant colony optimisation using the generalised assignment problem. In *2004 Congress on Evolutionary Computing*, pages 1916–1923, Portland, OR, USA, 2004.

[20] Y. Yang, G. Wu, J. Chen, and W. Dai. Multi-objective optimization based on ant colony optimization in grid over optical burst switching networks. *Expert Systems with Applications*, 37(2):1769–1775, 2010.

[21] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In X. Yao et al., editors, *Parallel Problem Solving from Nature, PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 832–842. Springer, 2004.

[22] E. Zitzler, L. Thiele, M. Laumanns, C. M.Fonseca, and da V. G. Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7:117–132, 2002.