An Efficient Experimental Methodology for Configuring Search-Based Design Algorithms

Simon Poulding, Paul Emberson, Iain Bate, and John Clark
Department of Computer Science, University of York, York, YO10 5DD, United Kingdom
E-mail: {simon.poulding,paul.emberson,iain.bate,john.clark}@cs.york.ac.uk

Abstract

Many problems in high assurance systems design are only tractable using computationally expensive search algorithms. For these algorithms to be useful, designers must be provided with guidance as to how to configure the algorithms appropriately. This paper presents an experimental methodology for deriving such guidance that remains efficient when the algorithm requires substantial computing resources or takes a long time to find solutions. The methodology is shown to be effective on a highly-constrained task allocation algorithm that provides design solutions for high integrity systems. Using the methodology, an algorithm configuration is derived in a matter of days that significantly outperforms one resulting from months of 'trial-and-error' optimisation.

1 Introduction

Designers of high assurance systems are required to solve problems known to be computationally hard. Search-based software engineering (SBSE) is a set of novel solution methods that apply search algorithms to solve such problems [3, 8]. Some of the first research into SBSE considered problems in high integrity systems engineering [20], and the search algorithms used by SBSE, such as Genetic Algorithms [9], Genetic Programming [12] and Simulated Annealing [11], have proven effective in solving the highly-constrained problems typical of this domain [7, 16, 22, 23]. Today, SBSE is applied to a wide range of problems throughout the development lifecycle, including requirements prioritisation [1], architectural design [14] and test data generation [13], and the field is growing rapidly.

As for any solution method, system designers—the end-users of the search algorithms—must be given convincing, problem-specific guidance that enables the effective use of SBSE if these methods are to be widely adopted. This is particularly important when solving complex problems, which by their nature, require a great deal of computational effort: the guidance must enable designers to configure algorithms to solve problems within practical time and resource limits.

However, search algorithm configurations are often tuned using a few 'trial-and-error' runs on an unrepresentative selection of test cases. The resulting choices can be far from optimal, causing the algorithm to take an excessively long time to locate solutions, or in the worst case, failing to locate a solution at all. But when each run of the algorithm requires large scale computational resources, or takes days to complete, a more rigorous determination of the optimal algorithm configuration can be challenging.

This paper presents an experimental methodology to be used in these situations. It enables the most effective algorithm configuration to be determined in a principled and resource-efficient manner, even for algorithms that are computationally expensive. When applied to a particular problem domain, the methodology provides the guidance that the system designer requires.

The methodology is demonstrated using a problem from high integrity systems engineering: that of solving task allocation problems of the type that arise in Integrated Modular Avionics (IMA) [10]. IMA design choices that ensure system integrity, such as hardware partitioning to separate critical and non-critical subsystems, impose additional constraints on the task allocation problem that make such problems difficult to solve. The experimental methodology is applied to a search algorithm that has proven effective in solving this type of task allocation problem through the use of a multi-component cost function.

The next two sections of the paper describe the task allocation problem and provide an overview of the search algorithm. Section 4 presents the experimental method. The experimental results are discussed in section 5. The paper concludes with an evaluation of the effectiveness of the experimental techniques, and proposes future work.

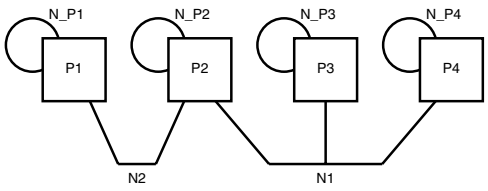


Figure 1. Example hardware architecture.

2 Task Allocation Problem

The problem is to find an allocation of tasks to processors so that timing requirements and resource constraints are met. As well as allocation, this can also involve assigning suitable task attributes such as priorities. For systems where tasks must communicate with messages, an allocation of messages to networks must also be found along with suitable message attributes.

The model defined here is a set of processors connected with bi-directional network links. Each link has a communication speed and latency which determines a message's communication time along with its size. All processing nodes have a high-speed link to themselves which simulates a communication mechanism for tasks on the same processor. Fig. 1 shows four processing nodes connected with two network buses and an additional four links for intra-processor communication.

Each schedulable object has a set of properties and timing requirements. For a task, these are worst case execution time (WCET), period and deadline. A message has a worst case size attribute, a source task from which it inherits its period and a destination task from which its deadline is derived. The dependencies introduced by messages form a directed acyclic communication graph. All of the objects in a single graph form a *transaction*. A system may have multiple transactions.

Using the task and message requirements given, a *configuration* consisting of an allocation to a *scheduler* (a processor or network) and a priority for each object must be found.

Once a candidate configuration has been established, the input requirements for tasks and messages can be used along with its configuration to perform scheduling analysis. A worst case communication time is calculated for each message based on the speed and latency of its scheduler in the configuration. The scheduling analysis needs to take account of precedence constraints between schedulable objects across multiple processors. For fixed priority scheduling, the analysis provided by [17] uses the concept of jitter to handle the varying delay caused by preceding objects. The response times provided by the analysis can be compared to deadlines to give a basis on which to calculate a quality metric for a configuration.

3 Search Method

The solution method uses the simulated annealing meta-heuristic search technique [11] which has been used previously for solving task allocation problems [20]. The algorithm makes small adjustments to a potential solution and evaluates each solution for its quality. The quality measurement is given in terms of a cost with the aim being to find the solution which produces the minimal cost value. A neighbourhood function generates a small change to the current configuration to move to the next one. This function randomly chooses a task or message and then performs a change to either its allocation or priority.

The cost function f is written as

$$f = \boldsymbol{g} \cdot \boldsymbol{w} \tag{1}$$

where $\boldsymbol{g} = (g_1, \ldots, g_n)^T$ is a vector of cost component functions and $\boldsymbol{w} = (w_1, \ldots, w_n)^T$ is a vector of real-valued weightings. The range of all cost components, g_i, is $[0, 1]$. The weightings are subject to the equality:

$$\sum_{i=1}^{n} w_i = 1 \tag{2}$$

Problem constraints can be treated as hard or soft by the search algorithm regardless of whether they are a hard or soft requirement. That is, the search can be allowed to evaluate a potential solution which breaks a hard constraint but will receive negative feedback from a cost component which indicates to what degree a constraint has been broken. Alternatively, the search can be prevented from considering these solutions altogether. There are advantages to the former. There may be no solution which meets all hard constraints in which case a best-effort solution can still be produced. This gives insight to the engineer into how requirements need to be adjusted or where more resources need to be provided. It can be computationally expensive to calculate a search move which does not break any constraints, but cheap to evaluate the component which indicates whether a constraint has been broken for a particular solution.

For the overall goal of achieving a schedulable solution, there are sub-goals which a human would intuitively use if they were to solve the problem by hand. An example of this is that all timing constraints cannot be met if any processor is too highly utilised. Therefore, apart from exceptional cases, a human would not consider a solution with all tasks on the same processor. Similarly communication precedence constraints cannot be met if communicating tasks are assigned to processors with no network link between them. An automated search can be given similar sub-goals. If the landscape defined by the cost function contains large

plateaux, a search algorithm cannot distinguish between solutions and will perform badly. The complex nature of task allocation problems in general, and the constraints specific to individual problem domains, will lead to a large number of cost function components being put forward for top level requirements, their sub-goals and for penalising constraint violations. However, the process of establishing how components affect search performance and solution quality, and also the comparative importance of different components, is often left to trial-and-error or ignored altogether.

The following sections describe an experimental methodology that was used to systematically evaluate a set of cost components for task allocation. The components are described in detail in the appendix.

4 Experimental Method

4.1 Objectives

The primary objective of the experiments was to make a principled determination of the optimal weightings for the components that make up the cost function (1). Optimality is defined for this purpose as those weightings that give the best performance, measured in terms of number of cost function evaluations taken by the algorithm to discover a schedulable task allocation, and considered over a defined range of problems instances.

However, initial research suggested that a single run of the algorithm could be computationally expensive: some weighting vectors took more than two days to find a schedulable allocation on a high-performance PC. The algorithm therefore presented an opportunity to investigate experimental techniques that made efficient use of computing resources. Three such techniques were identified: a strategy based on response surface methodology, the use of censoring to avoid long-running experiments, and an efficient experimental design to explore the response surface.

4.2 Strategy

The experimental strategy is a form of *response surface methodology* [15], where the *response* is the number of cost function evaluations. The algorithm has an expected response for each weighting vector, and the responses considered across all possible weighting vectors form a response surface. The location of the lowest point on this surface defines the optimal weightings.

By measuring the performance of the algorithm at well-chosen sample points, an equation is derived that models the response surface. The optimal weightings are then found by optimising the *model* in place of the algorithm. Computationally, algorithm runs are very expensive compared to the optimisation of the model equation, and so this strategy makes efficient use of available computing resources.

4.3 Censored Observations

It was observed that some algorithm runs took two days or more to complete, while others were significantly shorter, sometimes taking only a few minutes. The long-running experiments require a relatively large proportion of the available computing resources, but contribute little information about the response surface in the region of the optimal weightings. If, instead, long-running experiments were terminated when they reached a pre-determined number of evaluations, then many more experiments could be run with the same resources and more information gained about the response surface.

The technique of measuring responses only as far as a chosen threshold is termed *censoring*. The censored experiments are not 'wasted': they continue to provide some information—that the observed response would have been larger than the censoring threshold—but not as much as if they were allowed to continue until completion. However, the savings in terms of computing resources enables significantly more experiments to be run in total.

Censoring is widely used in the analysis of industrial engineering data [2] and medical trials [4] where the expected event (e.g. component failure) may not occur during the course of a time-limited experiment. Two models for censored observations that are often used in these fields, the Tobit model and Cox's Proportional Hazards model, were applied to the algorithm so that fit of the models to the actual response surface could be compared. Both of these censoring models incorporate a linear model. The linear model and the two censoring models are described below.

4.4 Mixture Model Canonical Forms

Standard linear models overparameterise the model since the weightings must sum to one (2).

One solution to this overparameterisation is to use the mixture canonical forms pioneered by Scheffé [19]. These forms are suitable as *mixture models* where the factors are constrained by an equality of the form of the weighting constraint. Since the second-order form is able to model curvature in the response surface [15], this was the form chosen for these experiments:

$$M(\boldsymbol{w}) = \sum_{i=1}^{9} \beta_i w_i + \sum_{i=1}^{9} \sum_{j=i+1}^{9} \beta_{ij} w_i w_j \qquad (3)$$

where $\boldsymbol{w} = (w_1, \ldots, w_9)$ is the weighting vector, β the model parameters to be determined by the experiments, and $M(\boldsymbol{w})$ the output of the linear model used to predict the response.

Table 1. Mean and variance of the algorithm response at three sample weighting vectors.

Weightings	$\overline{Y}/10^5$	$\mathrm{var}\,(Y)/10^{10}$	$\mathrm{var}\,(\log Y)$
$\boldsymbol{w}_a$	0.1366	0.0032	0.1697
$\boldsymbol{w}_b$	0.1679	0.0068	0.2407
$\boldsymbol{w}_c$	1.6930	1.4415	0.6856

4.5 Tobit Model

The Tobit model [21] incorporates a linear model in conjunction with an error term, U:

$$Y(\boldsymbol{w}) = M(\boldsymbol{w}) + U \tag{4}$$

where $Y(\boldsymbol{w})$ is the response and $M(\boldsymbol{w})$ the mixture canonical form (3).

The Tobit model assumes that U is a random variable that has a normal distribution with zero mean and constant variance σ^2, and that the value of U for each observation is independent of the others. The error term is used to account for the deviance of an individual observed response from the 'average' expressed by $M(\boldsymbol{w})$. In the context of the task allocation algorithm, this deviation is due to the stochastic nature of the simulated annealing method: if the algorithm were run multiple times using the *same* weighting vector, the observed responses would differ owing to random initialisation, the random selection of neighbouring solutions at each iteration of the algorithm, and the probabilistic acceptance of moves to solutions with a worse cost value.

There is no reason why the stochastic nature of the task allocation algorithm should exhibit the probability distribution assumed by the Tobit model. To assess the applicability of the Tobit model, some initial experiments were performed: for each of three weighting vectors, the algorithm was run 40 times on the same problem instance. The mean, $\overline{Y}$ and variance, $\mathrm{var}\,(Y)$, of the responses are shown in Table 1. The third column shows clearly that the variance of the response is not constant. However, if the response is modified to be the *logarithm* of the number of evaluations, the final column shows that this transformation gives a more reasonable approximation to a constant variance. Fig. 2 illustrates the same modified response (for $\boldsymbol{w_c}$) plotted against a theoretical normal distribution. The relationship is close to linear, especially in the centre of the range, indicating that the normal distribution is approximately satisfied by the logarithm of the number of evaluations.

Based on these results, the Tobit model of (4) is modified to become:

$$Y(\boldsymbol{w}) = \exp\{M(\boldsymbol{w}) + U\} \tag{5}$$

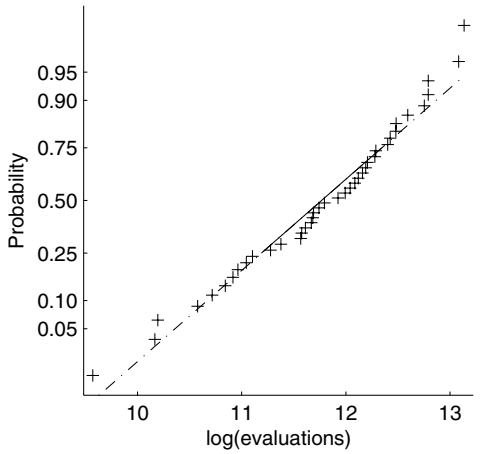


Figure 2. Normal probability plot of the logarithm of the number of cost function evaluations.

4.6 Cox's Proportional Hazards Model

The second censoring model is Cox's Proportional Hazards model [5]. This differs from the Tobit model in that it considers not the response, but instead the *hazard function*. If the point at which the algorithm solves a task allocation problem is considered as the 'event' of interest, then the hazard function, $h(e)$, at the number of evaluations, e, is a measure of the probability of this event occuring at e given that it has not occurred already. (The hazard function is often used for responses measured in continuous time, but is applied here with the discrete number of evaluations being the equivalent of the time variable.)

Cox's model assumes that the factors affect the hazard function multiplicatively so that:

$$h(e; \boldsymbol{w}) = \psi(\boldsymbol{w}) h_0(e) \tag{6}$$

where $\psi(\boldsymbol{w})$ is a chosen function of the factors, $h(e; \boldsymbol{w})$ the hazard function for the weighting vector $\boldsymbol{w}$, and $h_0(e)$ the *baseline hazard function*. The function $\psi(\cdot)$ is often taken to be a log-linear model [6], resulting in the following model of the hazard function:

$$h(e; \boldsymbol{w}) = \exp\{M(\boldsymbol{w})\}\, h_0(e) \tag{7}$$

No error term is necessary here to account for the stochastic nature of the algorithm. Cox's model is already probabilistic since $h(e)$ gives the *probability* of reaching a schedulable allocation at e number of evaluations (given that one has not yet been found).

An advantage of Cox's model compared to the Tobit model is that no significant assumptions are made as to the form of the hazard function and, correspondingly, as to the probability distribution of the response.

Table 2. Problem instance characteristics.

Characteristic	Valid Values
Number of tasks	10 – 75
Tasks per processor	5 – 15
Deadline	Period
Average processor utilisation	20% – 70%
Comm. graph longest path	50% – 100% no. tasks
Number of messages	100% – 250% no. tasks
Proportion of indep. tasks	0% – 40%

4.7 Problem Instances

A test case generator was used to create random problem instances. The parameters of the generator were set to produce problems whose characteristics covered the ranges listed in Table 2. The ranges were chosen to be representative of a medium-sized system, and to be wide enough to avoid overfitting the optimal weightings to a small set of similar problem instances. Instances were generated by randomly selecting problem characteristics from a uniform distribution over each range.

Some problem instances will be 'harder' than others to schedule, requiring the algorithm to make more cost function evaluations before a schedulable allocation is found. Although it would be possible to fit the above models to the 'average' response across a test set of random problem instances, the model would be less accurate when considered for a single problem instance, and the calculation of the 'average' response would need to accommodate censored observations. Instead, problem difficulty is incorporated by extending the linear model to include:

$$L(\boldsymbol{p}) = \sum_{k \geq 2} \alpha_k p_k \tag{8}$$

where p_k are problem indicator variables (set to 1 for problem k, otherwise 0), and α_k are additional model parameters to be estimated. $L(\boldsymbol{p})$ is added to $M(\boldsymbol{w})$ in the Tobit (5) and Cox's (7) models, so that $M(\boldsymbol{w})$ estimates the response for problem 1, and $L(\boldsymbol{p})$ adjusts the response to match the difficulty of problems 2 and above. By avoiding linear terms that combine w_i and p_k, the model can still be used to derive optimal weightings that are independent of the particular problem instance.

4.8 Experimental Design

A *simplex-lattice design* [19] for mixtures is used for the experimental design, the set of sample weighting vectors at which the response is measured. This design accommodates the weighting constraint (2) and is effective for fitting of the mixture canonical form model [19]. At sample points in this design, each weighting, w_i, is a multiple of $\frac{1}{m}$ where m is a chosen integer.

The design creates many sample points where some of the weightings are zero. Geometrically, these points are on the 'edge' of the response surface. It is still possible to find optimal weightings in the interior of the response surface using a design with many edge points, but it is necessary to assume that the model is a good fit for the response surface over the entire space: both at the edges and in the interior.

To test this assumption, a *constrained mixture design* is also used. This design imposes a lower bound, B_L, on the weightings to force sample points into the interior. To derive this design, a simplex-lattice design is first constructed on variables η_i, from which w_i derived using the formula:

$$w_i = B_L + (1 - 9B_L)\eta_i \tag{9}$$

Table 3. Experimental configuration.

Setting	A	B	C
Censoring Threshold	10^5	10^5	10^6
Simplex-Lattice Design m	3	3	3
Problem Instances	20	20	20
Weighting Lower Bound	0	0.05	0.01

4.9 Experimental Configuration

Three experiments were configured as shown in Table 3. Each experiment ran the algorithm against each of 20 randomly generated problem instances, at each of the chosen sample weighting vectors. A simplex-lattice design with $m = 3$ creates 165 sample weighting vectors, so a total of 3300 algorithm runs were required per experiment. Experiment A used an unconstrained design, while B represents a constrained design with a lower bound. The censoring threshold for A and B was chosen so that around half of the responses were censored. A higher threshold was set for experiment C to assess the accuracy of the censoring models. This experiment also used a constrained design with a lower bound closer to the response surface edge.

4.10 Analysis Method

The traditional method for estimating the parameters of both the Tobit and Cox's models is maximum likelihood estimation since censored observations can be incorporated easily into the likelihood function. This technique calculates the model parameters that would be most likely to have given the observed responses. For the Tobit model, the analysis also derives the variance of the error term, and for Cox's model, estimates the baseline hazard function.

Once a model has been fitted to the observed responses, the model is optimised to locate the weighting vector that gives the best algorithm performance. For the Tobit model (5), the optimal weightings are those that *minimise* the expected number of evaluations, Y. Since $\exp(\cdot)$ is a monotonically increasing function, it is sufficient to find the weightings that minimise the linear model $M(\boldsymbol{w})$. However, for Cox's model (7), the optimal weightings are those that have the highest hazard function, i.e., the highest probability of locating a schedulable solution at a given iteration of the algorithm. In this case, it is necessary to *maximise* $M(\boldsymbol{w})$.

In both cases, the linear model was optimised using an implementation of Sequential Quadratic Programming [18]. This an efficient optimisation method that can accommodate the weighting constraint (2). For experiments with a constrained design, the weighting lower bound, B_L, was applied as an additional constraint so that optima were found only within the region covered by the design. Since this optimisation method requires a seed solution, it was run 1,000 times with differently randomly generated seeds, and the best solution over the runs was taken as the estimate of the optimal weightings.

5 Results and Discussion

5.1 Optimisation Results

The optimal weightings estimated for the three experiments are shown in Table 4. The additional results labelled C′ are discussed in section 5.2 below.

The results demonstrate consistency across experiments and between the two different censoring models. In general, optimal weighting vectors are those that distribute the majority of the weight between w_1 and w_7 in a ratio near $3:2$, with the remaining weightings at or near the lower bound.

The exception from this general pattern is Cox's model estimate for experiment A. However, further investigation showed a local optimum at $(0.619, 0, 0, 0, 0, 0, 0.381, 0, 0)$ which corresponds to a hazard function approximately 83% as large as that at the global optimum. The modelled response surfaces were often found to have more than one optimum such as those found in this case.

For the Tobit model, maximum likelihood estimation also returns the variance, σ^2, of the normal distribution of the error term, U. For experiments A and C the variance is 2.192 and 2.608 respectively. However, the variance estimated for experiment B is significantly lower at 0.751, and very similar to the values found during the initial research on the algorithm (last column of Table 1). The design for experiment B used sample points that were further away from the edge than the other two experiments, and the lower variance *may* indicate that the second-order mixture canonical form (3) is able to closely model the response surface over the interior of the surface. Conversely, the model is unable to accommodate a different behaviour sampled by weighting vectors close to or at the edge of the surface in experiments A and C. The lack of fit in these two experiments requires a higher variance in the error term in order to explain the observed responses.

Fig. 3 provides a visual respresentation of the fit of the model to the observed responses. Since the response surface is over a high-dimensional space of valid weighting vectors, it cannot be visualised directly as a surface. Instead, sample points are chosen on the model surface and the predicted response calculated at each point. The predicted and observed responses are plotted at the sample points, with points sorted in order of the predicted response. The sample points chosen were the same as for the corresponding experimental design as this is convenient for plotting the observed responses.

Fig. 3a demonstrates the Tobit model from experiment A using a problem instance whose 'difficulty' was in the middle of the range. Fig. 3b shows the fit for the same problem instance using the Tobit model from experiment B. For clarity over the large range of values, the *logarithm* of the response is plotted on the y-axis. The 95% confidence limits indicated by dotted lines are constructed by taking an interval $\pm 2\sigma$ around the predicted response, where σ^2 is the error term variance estimated by the analysis. The majority of the observed values fall within the 95% confidence interval for both experiments but the need for a larger confidence interval in experiment A can be seen by the greater spread of observed responses, especially for design points close to the optimum at the left of the figure.

It is also noticeable that many of the observed responses in experiment A are larger than those of experiment B, so much so that the responses reach the censoring threshold at the right of Fig. 3a. Since the sample points for experiment A are at the edge of the surface, and for B are slightly away from the edge, this difference suggests that the response of the algorithm is poorer at the surface edge compared to nearby points that are not at the edge.

5.2 Effect of Censoring Threshold

To evaluate the effect of the censoring threshold, the analysis of experiment C was repeated with the observed responses truncated at a threshold of 10^5. This simulated the same experiment with a lower censoring threshold in order to compare results.

The results for the lower threshold are labelled C′ in Table 4. The estimated optimal weightings are very close to those obtained from experiment C using a threshold 10 times as high.

Table 4. Optimal weightings estimated using (a) the Tobit model and (b) Cox's model. An asterisk indicates that the optimal weighting was the value of the lower bound B_L.

Expt	B_L	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
A	0	0.586	*	*	*	*	*	0.414	*	*
B	0.05	0.373	*	*	0.064	0.071	*	0.242	*	*
C	0.01	0.578	*	*	*	*	*	0.352	*	*
C′	0.01	0.571	*	*	*	*	*	0.359	*	*

(a)

Expt	B_L	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
A	0	0.303	*	0.185	0.302	0.201	*	*	*	*
B	0.05	0.389	*	0.061	*	0.100	*	0.200	*	*
C	0.01	0.602	*	*	*	*	*	0.328	*	*
C′	0.01	0.598	*	*	*	*	*	0.332	*	*

(b)

The effect on the model response surface is shown in Fig. 4. The dashed line shows the response predicted using the original threshold (C), and the full line, using the lower threshold (C′). (The jaggedness of the C′ line arises from the arbitrary choice of the experiment C responses for the ordering of the sample points. It is not indicative of a more uneven response surface.) The figure suggests good correspondence between the two fitted response surfaces, especially for small responses close to the optimum at the left of the graph. The lower threshold model appears to underestimate the response at higher values in comparison to the original threshold.

With the original threshold, approximately 33% of the observations were censored; with the lower threshold, this rises to 55%. Using the original threshold, a total of approximately 1.44×10^9 evaluations were made across all experiments. If the simulated lower threshold had been used, only 2.00×10^8, or 13.9% of these evaluations would have been made. Of course, the number of evaluations that would have been made if *no* censoring had occurred would be even higher, and the savings made by using the low threshold of experiment C′ would have been even greater.

These results demonstrate that a low censoring threshold can greatly reduce the computational expense of the experiment, while retaining accurate results for both the optimal weightings and the model of the response surface.

5.3 Verification Experiment

To verify the optimal weightings estimated by the models, the four weighting vectors shown in Table 5 were applied to 100 randomly generated problem instances (of which 20 were the instances used in experiments A to C). D1 and D3 are representatives of the optimal weightings found using unconstrained (experiment A) and constrained (experiment B) designs, respectively. D2 is a point lying between D1 and D3 close to, but not at the edge of the response surface. Points D2 and D3 are motivated by the results above indicating that design points slightly away from the edge appear to show better performance than equivalent points on the edge. D4 is an optimal weighting vector found by trial-and-error over many months prior to the principled experimentation described in this paper, and acts as a comparison point.

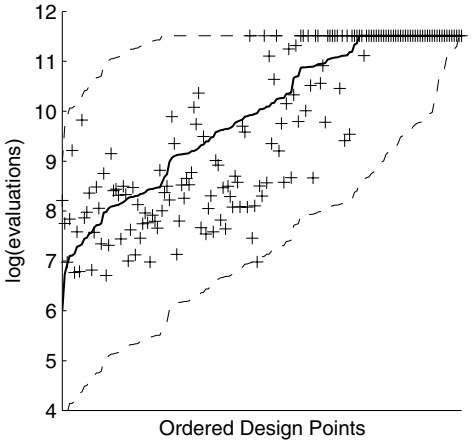
In these experiments there was no need to censor results. This permitted the performance to be compared using the median response of each experiment, supplemented by a nonparameteric paired-sample Wilcoxon signed rank test [24] to test that differences in the medians were significant at the 5% level.

All tests gave the same ordering: the algorithm performs best at the D2 weighting vector, followed by D3, D4 and D1 in order. A comparison of the medians showed that D2 would, on average, require 33% less evaluations than the trial-and-error derived weightings of experiment D4.

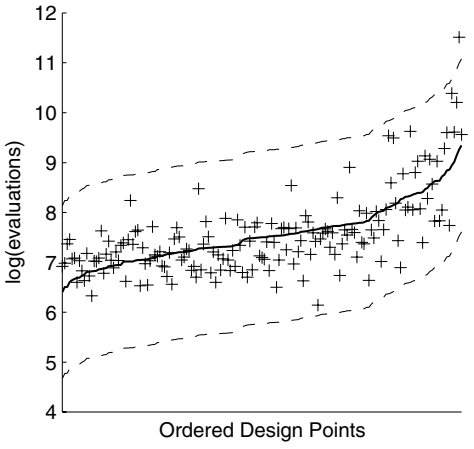
Against expectation, the unconstrained optimal weightings (D1) were significantly *worse* than the manually derived weightings, while weightings very close to these values, but away from the edge (D2) are significantly better. This provides further evidence that the behaviour of the response surface close to the edge changes rapidly and differs from that in the interior of the surface: the response appears to improve closer to the edge (D2 is better than D3), then quickly gets worse right at the edge itself (D1 is very much

Table 5. Weighting vectors compared by the verification experiment.

Expt	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
D1	0.6	0	0	0	0	0	0.4	0	0
D2	0.596	0.001	0.001	0.001	0.001	0.001	0.397	0.001	0.001
D3	0.405	0.05	0.05	0.05	0.05	0.05	0.245	0.05	0.05
D4	0.159	0.159	0.318	0.042	0.027	0.042	0.053	0.042	0.159



(a)



(b)

Figure 3. Observed responses (logarithm of the number of evaluations) at design points for (a) experiment A, (b) experiment B. The solid line is the mean response predicted by the Tobit model, and the dashed lines mark the 95% confidence interval.

worse than D2). The second-order mixture canonical form (3) would be unable to accurately model such 'edge effects'.

A possible explanation for the change of behaviour close to the response surface edge is provided by consideration of

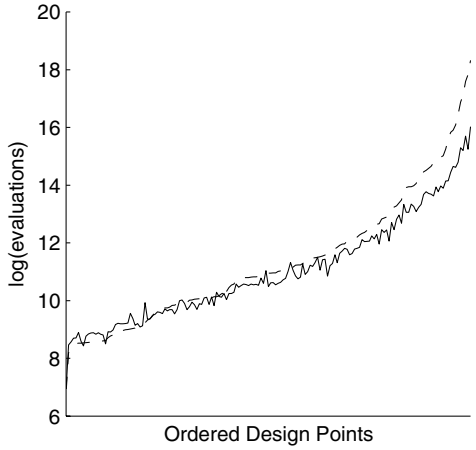


Figure 4. Predicted response of models derived from experiments C (dashed line) and C′ (full line).

the cost function formed using these weightings. The components corresponding to weightings w_1 and w_7 appear to provide the most advantage to the cost function in guiding the algorithm to a schedulable allocation. However, all the components provide some guidance. So when components g_1 and g_7 give the same cost function value for a number of possible moves by the algorithm, the small contribution of the remaining components enables the algorithm to break such ties in a helpful manner. In terms of the cost function landscape, the small values contributed by the other components avoid plateaux and ensure a guiding gradient (however small) throughout the landscape.

6 Conclusion

The experiments have met their primary objective of finding an optimal cost function for the task allocation algorithm. The verification experiment demonstrated that the optimal weightings resulted in an algorithm that significantly outperformed the weightings that had previously been identified by 'trial-and-error'. Since the solution of the complex task allocation problem described here can require a great deal of computation, this improvement represents significant cost and time savings for system designers.

The efficiency of the experimental methodology has been demonstrated. The 'trial-and-error' best weightings had been located over a number of months of investigation, while the optimal weightings were found in only a few days using the experimental methodology. The use of a relatively low censoring threshold, in particular, has reduced the number of cost function evaluations required to model the response to under a seventh of that required with a high threshold, without a significant loss of accuracy.

The optimal configuration of the algorithm will be used as a baseline for further investigation of the robustness of task allocation solutions and the scalability of the solution method. Both these properties will increase the complexity of the problems; scale through the number of tasks and messages, and robustness by placing additional constraints on the solutions. It is likely that finding a schedulable solution to these problems will require even more computational effort, and the experimental methodology described in this paper will be vital in configuring efficient algorithms for this search.

Another objective of future research is to use models that, firstly, describe the behaviour of the algorithm close to the edge of the response surface, and secondly, enable problem-specific optimal weightings to be derived from problem characteristics.

This work has demonstrated that principled guidance on algorithm configuration can be derived for system designers in a computationally-efficient manner. The future research on robustness, scalability and improved response surface models promises to refine this guidance and expand the problem domains to which the methodology can be applied.

Appendix - Cost Function Components

The sets of tasks, messages, processors and network links are written as $\mathcal{T}$, $\mathcal{M}$, $\mathcal{P}$, and $\mathcal{N}$. The set of objects contained in transaction r is TRANS$_r$. The set of all transactions is TRANS. $\mathcal{S} = \mathcal{T} \cup \mathcal{M}$. The number of elements in a set X is denoted as $|X|$. Directly dependent (DD) tasks are a pair of tasks which have a message sent between them. Indirectly dependent tasks (ID) appear in the same transaction but are not necessarily adjacent.

The first component assesses the number of unschedulable tasks and messages.

$$h_1(\tau) = 1 \text{ if } R_\tau > D_\tau \text{ else } 0 \tag{10}$$

$$g_1 = \frac{1}{2|\mathcal{T}|}\sum_{\tau\in\mathcal{T}} h_1(\tau) + \frac{1}{2|\mathcal{M}|}\sum_{\rho\in\mathcal{M}} h_1(\rho) \tag{11}$$

The following component counts how many DD tasks are allocated to unconnected schedulers. Let al map an object to its allocated scheduler and V map a scheduler to the set of schedulers to which it is connected.

$$c(\tau, \upsilon) = 1 \text{ if } V(\mathrm{al}(\tau)) \cap V(\mathrm{al}(\upsilon)) = \emptyset \text{ else } 0 \tag{12}$$

$$g_2 = \frac{1}{|\mathcal{M}|}\sum_{\rho\in\mathcal{M}} c(\mathrm{src}(\rho), \mathrm{dest}(\rho)) \tag{13}$$

The following component penalises objects which cannot receive their input or send their output due to an invalid allocation.

$$\mathrm{in}(\tau) = \begin{cases} \{\rho \in \mathcal{M} : \mathrm{dest}(\rho) = \tau\} \text{ if } \tau \in \mathcal{T} \\ \{\mathrm{src}(\tau)\} \text{ if } \tau \in \mathcal{M} \end{cases} \tag{14}$$

$$\mathrm{out}(\tau) = \begin{cases} \{\rho \in \mathcal{M} : \mathrm{src}(\rho) = \tau\} \text{ if } \tau \in \mathcal{T} \\ \{\mathrm{dest}(\tau)\} \text{ if } \tau \in \mathcal{M} \end{cases} \tag{15}$$

$$g_3 = \frac{1}{2|\mathcal{S}|}\sum_{\tau\in\mathcal{S}}\Big[\frac{|\{\upsilon\in\mathrm{in}(\tau):\mathrm{al}(\upsilon)\notin V(\mathrm{al}(\tau))\}|}{|\mathrm{in}(\tau)|} + \frac{|\{\upsilon\in\mathrm{out}(\tau):\mathrm{al}(\upsilon)\notin V(\mathrm{al}(\tau))\}|}{|\mathrm{out}(\tau)|}\Big] \tag{16}$$

The following component measures priority assignment which are incompatible with precedence constraints. $\mathrm{pre}(\tau)$ is the set of all objects preceding τ and $\mathrm{post}(\tau)$ is the set of all objects that follow τ.

$$g_4 = \frac{1}{|\mathcal{S}|}\sum_{\tau\in\mathcal{S}}\Big[\frac{|\{\upsilon\in\mathrm{pre}(\tau) \text{ and } P_\upsilon > P_\tau\}|}{|\mathrm{pre}|(\tau)} + \frac{|\{\upsilon\in\mathrm{post}(\tau) \text{ and } P_\upsilon < P_\tau\}|}{|\mathrm{post}(\tau)|}\Big] \tag{17}$$

A sensitivity component calculates the largest factor by which execution/communication times can be scaled and for the system to be schedulable. This value can be found using a binary search and will be less than 1 while the system is unschedulable.

$$g_5 = e^{-\lambda \mathrm{SCAL}_\mathcal{S}} \tag{18}$$

where SCAL$_S$ is the largest value of a scaling factor s such that the system is schedulable when the WCETs, C_i of objects in the set $\mathcal{S}$ are set to $\lfloor sC_i \rfloor$.

The load balancing component is based upon the variance of the utilisations of processors:

$$g_6 = \sqrt{\frac{\sum_X (U_X - \mu^2)}{(|\mathcal{P}|-1)\mu^2 + (U-\mu)^2}} \tag{19}$$

where U_i is the utilisation of processor i, and μ is the mean utilisation.

Grouping objects reduces overheads. V_r is the set of tasks in TRANS$_r$. For each $\tau_i \in V_r$, the number of tasks allocated to the same scheduler as τ_i and also in V_r is a_{ri}. A grouping value and its maximum for each transaction is:

$$\gamma = |V_r| - \sum_i \frac{a_{ri}}{|V_r|} \qquad \gamma_{\mathrm{MAX}} = \frac{|V_r|(|\mathcal{P}|-1)}{|\mathcal{P}|}$$

Similar formulae can be defined for messages with W_r being all messages in TRANS_r. Using γ_{MAX} to normalise γ and then summing over all transactions, the component formula is

$$g_7 = \frac{(|\mathcal{P}|-1)}{2|\text{TRANS}||\mathcal{P}|}\sum\nolimits_r \left(|V_r|^2 - \sum\nolimits_i a_{ri}\right) + \frac{(|\mathcal{N}|-1)}{2|\text{TRANS}||\mathcal{N}|}\sum\nolimits_r \left(|W_r|^2 - \sum\nolimits_i a_{ri}\right) \quad (20)$$

Component g_7 groups ID tasks but messages between DD tasks may still go back and forth between processors. The following penalises messages sent between DD tasks on different schedulers.

$$g_8 = \frac{1}{|\mathcal{T}|}\sum_{\tau\in\mathcal{T}}\left[\frac{|\{\rho\in\text{in}(\tau):\text{al}(\text{src}(\rho))\neq\text{al}(\tau)\}|}{|\text{in}(\tau)|} + \frac{|\{\rho\in\text{out}(\tau):\text{al}(\text{dest}(\rho))\neq\text{al}(\tau)\}|}{|\text{out}(\tau)|}\right] \quad (21)$$

An additional penalty is given to solutions containing schedulers with over 100% utilisation.

$$g_9 = \frac{|\{l\in\mathcal{P}\cup\mathcal{N}:U_l>100\}|}{|\mathcal{P}\cup\mathcal{N}|} \quad (22)$$

Acknowledgements

This work was funded by Engineering and Physical Sciences Research Council grant EP/D050618/1, SEBASE: Software Engineering by Automatic SEarch.

The authors would also like to thank their colleagues who donated processing time for use in these experiments.

References

[1] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *ICSM*, pages 176–185. IEEE Computer Society, 2006.

[2] A. H. Chowdhury and N. S. Fard. Estimation of dispersion effects from robust design experiments with censored response data. *Quality and Reliability Engineering International*, 17(1), 2001.

[3] J. Clark, J. J. Dolado, M. Harman, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, and M. Roper. Reformulating software engineering as a search problem. In *IEE Proceedings Software*, 2003.

[4] D. Collett. *Modelling survival data in medical research*. CRC Press, 2nd edition, 2003.

[5] D. R. Cox. Regression models and life tables. *Journal of the Royal Statistical Society, Series B*, 34:187–220, 1972.

[6] D. R. Cox and D. Oakes. *Analysis of Survival Data*. Monographs on statistics and applied probability. Chapman and Hall, 1984.

[7] P. Emberson and I. Bate. Minimising task migrations and priority changes in mode transitions. In *Proceedings of the 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 07)*, pages 158–167, 2007.

[8] M. Harman and B. F. Jones. Search-based software engineering. *Information & Software Technology*, 43(14):833–839, 2001.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[10] P. Hollow, J. McDermid, and M. Nicholson. Approaches to certification of reconfigurable IMA systems. In *Proceedings 10th International Symposium of the International Coucil on Systems Engineering*, 2000.

[11] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[12] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

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

[14] B. S. Mitchell, S. Mancoridis, and M. Traverso. Using interconnection style rules to infer software architecture relations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004.

[15] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, Inc., 5th edition, 2001.

[16] M. Nicholson. Supporting design of safety-critical systems. In *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 202–207, July 2003.

[17] J. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[18] M. J. D. Powell. Variable metric methods for constrained optimization. In A. Bachem, M. Grotschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 288–311. Springer Verlag, 1983.

[19] H. Scheffé. Experiments with mixtures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):344–360, 1958.

[20] K. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[21] J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica*, 26:24–36, Jan 1958.

[22] N. Tracey. *A Search Based Automated Test-Data Generation Framework for Safety-Critical Systems*. PhD thesis, University of York, 2000.

[23] T. Wiangtong, P. Y. Cheung, and W. Luk. Comparing three heuristic search methods for functional partitioning in hardware-software codesign. *Design Automation for Embedded Systems*, 6(4):425–449, July 2002.

[24] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.