The Influence of Search Components and Problem Characteristics in Early Life Cycle Class Modelling

J.E. Smith*, C.L. Simons

Department of Computer Science and Creative Technologies, University of the West of England, Bristol, UK

*Corresponding author

Email address: `james.smith@uwe.ac.uk` (J.E. Smith)

URL: `http://www.bit.uwe.ac.uk/~jsmith` (J.E. Smith), `http://www.bit.uwe.ac.uk/~clsimons` (C.L. Simons)

The Influence of Search Components and Problem Characteristics in Early Life Cycle Class Modelling

J.E. Smith*, C.L. Simons

Department of Computer Science and Creative Technologies, University of the West of England, Bristol, UK

Abstract

This paper examines the factors affecting the quality of solution found by meta-heuristic search when optimising object-oriented software class models. From the algorithmic perspective, we examine the effect of encoding, choice of components such as the global search heuristic, and various means of incorporating problem- and instance-specific information. We also consider the effect of problem characteristics on the (estimated) cost of the global optimum, and the quality and distribution of local optima.

The choice of global search component appears important, and adding problem and instance-specific information is generally beneficial to an Evolutionary Algorithm but detrimental to Ant Colony Optimisation. The effect of problem characteristics is more complex. Neither scale nor complexity have a significant affect on the global optimum as estimated by the best solution ever found. However, using local search to locate 100,000 local optima for each problem confirms the results from meta-heuristic search: there are patterns in the distribution of local optima that increase with scale (problem size) and complexity (number of classes) and will cause problems for many classes of meta-heuristic search.

Keywords: Search-Based Software Engineering, Class Modelling, Evolutionary Algorithms, Ant Colony Optimisation, Memetic Algorithms

*Corresponding author

Email address: `james.smith@uwe.ac.uk` (J.E. Smith)

URL: `http://www.bit.uwe.ac.uk/~jsmith` (J.E. Smith), `http://www.bit.uwe.ac.uk/~clsimons` (C.L. Simons)

1. Introduction

The task of class modelling within early cycle object orientated software engineering is often poorly tackled by humans. Issues such as scale and complexity pose significant issues, but the ongoing history of software failures shows that their relative effect in creating difficulties is not well understood. Recent research has demonstrated that this task, also referred to as the Class Responsibility Assignment Problem, can be successfully tackled by posing it as a search problem. For the sake of brevity we will hereafter refer to it as "class modelling", with the restriction to the context of the early stages of the development life cycle being taken as read. The "Search Based Software Engineering" (SBSE) approach to class modelling has been illustrated using both Evolutionary Algorithms (EA) and Ant Colony Optimisation (ACO) to perform the underlying search. Each of these outperforms methods based on a single improving solution, and has been shown to display strengths and weaknesses - both in terms of optimisation performance, and of how easily "standard" algorithms can be applied to the domain. However, three major questions remain unanswered. The first is whether the problems caused by scale and complexity are a result of human limitations, or do they also exist when the task is formulated for automated search? The second is how task-specific information can be incorporated at various levels to manage the global-local search trade-off, and aid search by avoiding breaking constraints. The third is the identification of design problem characteristics that make automated search harder, to inform the creation of a richer and more rigorous suite of benchmark problems than currently exists.

Our contention is that because of its complex, subjective nature, class modelling should be tackled via interactive search, augmented with a surrogate fitness function to prevent user fatigue. Therefore ideally the choice of search method should consider the ease with which it can support users' input via actions such as "freezing" satisfactory parts of designs. Previously [19, 20] we compared ACO and EAs as global search algorithms for this task, concluding that performance issues aside, there are practical reasons for preferring to use some heuristics. For example, in an ACO "freezing" of partial solutions can be simply achieved via direct changes to the pheromone table. In contrast, it would necessitate manipulation of an EA's recombination and mutation operators on-the-fly, although progress has been made in this direction by interleaving phases of human manipulation and evolution [27].

However, these considerations are orthogonal to the comparative search performance, so understanding the latter is vital before we can disregard it, and clearly algorithmic simplicity cannot be a substitute for poor performance. Initial investigations [20] showed that in their canonical form EAs outperformed ACO. In a recent paper we extended this work to incorporate the effect of local search to create two different examples of the class of memetic algorithms (MAs) [9]. Those preliminary studies still showed that the MAs using evolution as the global search component (hereafter M-EAs) found higher quality solutions than those based on ACOs (hereafter M-ACO). It should be noted that for the sake of "fairness" the ACOs in those papers did not make any use of heuristic or instance-specific information.

This paper extends that study to examine the effect of different ways in which information can be incorporated within meta heuristic search. Each of these creates its own bias in determining the probability distribution functions that govern the generation of candidate solutions. From the SBSE perspective it is important to gain an understanding of how these impact on performance as class modelling problems vary in scale and complexity. In order to provide some insights into these issues, Section 2 provides a brief background to previous research in this problem domain, and the relationship between class modelling and the more abstract problem of graph partitioning. Section 3 describes the chosen representation, the global and local search components considered, and different ways in which problem-specific information can be incorporated. Section 4 describes the experimental methodology used, then Section 5 describes, and Section 6 analyses the results obtained. Finally Section 7 summarises the findings and implications for SBSE in general, and early stage class modelling in particular.

2. Background

2.1. What Makes a Class Modelling Problem Hard?

Early lifecycle class modelling is an intensely human activity wherein relevant concepts and information relating to a design problem are identified. High quality class models are crucial as the basis of subsequent software development activities, as inferior designs can lead to deleterious and costly down-stream consequences. Starting from use case descriptions or user stories from the design problem domain, various required software system actions and data are identified. In the object-oriented paradigm such actions and data correspond to candidate 'methods' and 'attributes' to be grouped by means of the 'class' construct. Class models thus reveal how these groupings relate to relevant concepts and information in the problem domain. There is evidence to suggest that act of early lifecycle class modelling is non-trivial and demanding to perform, not least due to the scale and complexity of the problem domain. For many problems the number of methods and attributes can run to hundreds, with a corresponding multiplicity of classes. Petre [13] has suggested that software design problems are often wicked: too big, too ill-defined, too complex for easy comprehension and solution. Sometimes the problems are only fully understood after they are solved. Solving such problems is rarely a matter of brute force or routine. Glass [7] goes further, suggesting that the scale and complexities of some software designs may be beyond human comprehension. There is also evidence that designers are blessed with varying degrees of modelling talent. Even for experienced modellers, Glass notes that designer performance may vary from 28:1 from the best to the worst. Curtis [4] also notes a range in talent, observing that only super-designers can reason across the full breadth and depth of complex, ill-structured problems in order to fully consider the consequences and decisions of modelling decisions. From the field of education there is evidence that class modelling is difficult to learn. In a study of 740 undergraduates and 135 design problems, Svetinovic et al. [26] observe that with respect to concept identification, *"some students just don't get it"*.

To help overcome these difficulties, manual heuristics and search strategies are available. For example, Larman [10] suggests that the required actions of the software system-to-be be regarded as responsibilities, i.e., a contract or obligation of a class-to-be. Class modelling then becomes an exercise in assigning responsibilities to candidate classes. Larman proposes General Responsibility Assignment Software Patterns (GRASP) to guide the class modeller. GRASP reflect modelling principles such as separation of concerns, high internal class cohesion, and low coupling between classes. Wirfs-Brock and McKean [29] describe a responsibility-driven design approach to class modelling as a process of discovery and invention. They propose a manual search strategy, in which modellers make educated guesses about the kinds of inventions needed based on the nature of the problem domain and the things that are critical to it. In this manual search candidate models are evaluated from the perspectives of: information flowing through the model; decision making, control and coordination activities; and representations of real-world things that the model needs to know about.

2.2. Search-Based Software Engineering

Search-Based Software Engineering (SBSE) is a well-established discipline, applied across the whole software development lifecycle [30]. Historically, comparatively little focus has been directed to the upstream stages, although this is beginning to be addressed. Typically metrics relating to coupling and cohesion are used to guide meta-heuristic search of design spaces of object-oriented class models. Bowman *et al.* [2] used a multi-objective EA to optimise designs for a number of pre-specified metrics, but only considered a single problem instance. Simons and Parmee [18, 17] applied interactive EAs, using linear regression to learn a surrogate fitness model combining coupling with a number of "elegance metrics" to approximate users' subjective preferences for different problems. Working slightly later in the development life-cycle, Sievi-Korte *et al.*[15] used an M-EA, and Vathsavayi *et al.*[27] interleaved human and evolutionary adaptation of the usage of patterns.

Although any search algorithm could be used in SBSE, research effort has tended to concentrate on EAs. Previously we compared the use of ACOs and EAs for this problem [19, 20], concluding that given sufficient computational budget, global search via EAs was more effective at finding high quality solutions than that using ACO. When the computational budget was reduced (as is, for example, often the case in interactive search) the situation was reversed. However, with both algorithms, and both representations examined, a major issue was dealing with the constraint that a valid class model should contain at least one attribute and at least one method. Those papers used penalty functions (all invalid models were given zero fitness) and the use of random regeneration of invalid solutions. Dealing more efficiently with this constraint would necessitate either a significant adaptation of the underlying global search heuristics, or the provision of a "repair" mechanism.

A well designed Local Search algorithm that systematically examines the effect of moving elements between classes provides a simple way of providing the latter, and also of improving valid solutions. In a recent paper we reported preliminary studies on the influence of such a component [22]. Results showed that in the absence of other forms of information the M-EAs still discovered higher quality solutions that M-ACO, albeit over a longer timescale. This could be interpreted in one of two ways - either that the M-ACO are better given a limited computational budget, or that the M-EAs are better able to escape local optima. However, that paper did not permit any use of heuristic functions or other problem-specific adaptations. While such knowledge might require substantial alterations of operators within an evolutionary framework, it is standard practice in ACO research and is readily incorporated, as will be shown in later sections.

2.3. Relationship to Graph Partitioning and Similar Problems

The creation of class models can be thought of as a process of grouping elements (attributes and methods) into classes, so that every class contains at least one element of each type. The initial description of the problem as captured via, for example, use-cases, specifies a number of "uses". For example, in a sales application a method "create invoice" might make reference to attributes such as "customer-id". Therefore it is natural to think of the combination of elements and uses as defining a graph $G = (V, E)$ where the sets of methods M and attributes A are vertices (i.e., $V = M \cup A$) and a use of attribute $i \in A$ by a method $j \in M$ creates an edge $e_{ij} \in E$.

While many different metrics have been proposed to capture some measure of the quality of a software design, it is commonly recognised that a good design will exhibit high cohesion and loose coupling. Hence, uses tend to happen within a class, and the usage of attributes from one class by methods of another is minimised. In terms of a graph structure this means dividing the vertices into a number of discrete partitions so that the number of edges between partitions in minimised.

This description makes the link between class modelling and graph partitioning clear. Some differences exist - for example, graph partitioning methods assume all vertices are of the same type and in the most common form of "uniform graph partitioning" they explicitly try to minimise imbalance, usually with fixed number of partitions. Closely related "grouping" problems such as the multiple knapsack problems may consider the "cost" or "value" or a vertex, but still treat all as essentially of the same type. We note that requiring each class to contain at least one method and at least one attribute creates an extra constraint since not all vertices are of the same type.

Graph partitioning is NP-hard and has been extensively studied [1, 8]. A full review is beyond the scope of this paper, but it is notable that many successful recent applications use a form of spectral partitioning based on an adjacency graph (an alternative representation of the edge set), and repeatedly bi-partition the graph and then its components. In Section 3.6 we describe the creation of a distance matrix to be used by the ACO, based on a generalisation of adjacency matrices.

Both EAs and ACO [28] have been proposed for graph-partitioning and related problems. One major problem addressed by [6, 11] is that of representation. One active research question is whether it is possible, or indeed desirable, to avoid the explicit redundancy that comes from assigning arbitrary labels to groups. A further problem is that even multiple-colony ACO models, wherein each colony forms a separate partition, use a permutation type representation by default. This introduces extra redundancy, since the many solutions corresponding to different permutations of the nodes in a partition are semantically identical. Section 3.5 describes a modification of ACO's pheromone update mechanism to remove this redundancy.

3. Representation and Algorithms

This section briefly overviews the way in which class modelling is posed as a search problem, and the three search algorithms used in this paper.

3.1. Representation

In order to map readily onto freely available ACO source code, we borrow a representation commonly used for Vehicle Routing Problems. A solution is considered to be a permutation of a set of items, comprising the design elements plus "end-of-class" markers. If a problem has a attributes and m methods, to be grouped into up to c classes, the representation is then a permutation of size $l = a + m + c - 1$. The value in the i^{th} position denotes the element that occurs in the i^{th} place in the tour, where the attributes are numbered from 1 to a, the methods from $a+1$ to $a+m$ and the "end-of-class" markers from $a+m+1$ to $a+m+c-1$. Since the permutation for a candidate solution might contain adjacent end-of-class elements, c represents an upper limit on the number of classes in any given solution, rather than a fixed value. When the representation of a solution is decoded to give the candidate class model, a simple array $class = [c_1, \ldots, c_{a+m}]$ is constructed whose j^{th} element gives the class (from 1 to c) of element j using the numbering scheme above.

As discussed in the previous section, this representation contains two sources of redundancy: the order in which classes are specified; and the ordering within a class. The former is common to many meta-heuristic approaches to grouping problems, and it has been argued that it can be beneficial by increasing diversity and delaying convergence [11]. To investigate the effect of the latter, Section 3.5 investigates a mechanism for its removal.

3.2. Evolutionary Algorithm

The EA chosen is a standard generational genetic algorithm. Parents are chosen by deterministic binary tournaments, with elitism so that the incumbent replaces the least fit of the newly created offspring if better. Crossover was applied with probability 0.7. We do not provide a full algorithmic description of the EA as this is a well known algorithm with many descriptions online. Full details of the operators used can be found in [5]. Parameter values were determined by extensive preliminary testing. To examine the effect of in-class redundancy we investigated two different representations. The first was the permutation outlined above with length $l = a + m - 1$, in which case we used Edge Recombination, and mutation randomly applied one of *Swap*, *Insert* and *Invert* operators. The second was a direct encoding of class labels to elements with a representation of length $l = a + m$ and search space $\{1, \dots, c\}^{a+m}$. In this case we used uniform crossover, and mutation randomly chose a new allele value for each position applied.

There is now a significant body of evidence in favour of the use of self-adaptive mutation rates (see, for example, [12]). Following one of the schemes proposed by Serpell and Smith [14], each individual contained an extra gene encoding the mutation probability p_m, and we adopted the following method for mutation of each offspring:

1. With probability 0.2 the encoded value of p_m was changed to a new value, selected uniformly at random from the fixed set of allowed values.
2. The value for p_m was multiplied by the representation length l, and the result truncated to an integer to give the number of mutation events.
3. The chosen mutation operator was then applied the number of times determined in the previous step to randomly chosen positions.

A fixed set of ten mutation rates was defined as $p_m \cdot l \in \{0.001, 0.002, 0.01, 0.02, 0.1, 0.2, 1, 2, MIN(0.25l, 5), MIN(0.25l, 10)\}$. A range of population sizes were used as detailed in Section 4.4.

3.3. Ant Colony Algorithm

Following some preliminary experimentation, and guided by the findings reported in [19, 20], we applied a *MAX-MIN* ACO [25]. The following brief description considers a problem where the ACO seeks a least-cost path through a set of nodes, such as the TSP. Full details may be found in [24, 25].

The ACO maintains an $l \times l$ "pheromone matrix" P, which reflects the search history, and an $l \times l$ matrix H of heuristic information. Together these define the probability distribution function for the generation of new solutions. In each iteration N ants are each placed at random at a starting node and tours are independently created. To construct its tour, at each successive node i, the ant creates a list S of all the as-yet unvisited nodes. It then selects a next node $j \in S$ from the list with probability:

$$p_{move}(ij) = \begin{cases} \frac{P_{ij}^{\alpha} \cdot H_{ij}^{\beta}}{\sum_{j \neq i, j \in S} P_{ij}^{\alpha} \cdot H_{ij}^{\beta}} & \text{, if } j \in S \\ 0 & \text{, otherwise.} \end{cases} \tag{1}$$

After the ant has constructed a full solution its fitness f is measured. If *best* denotes the least cost path for generation t, and $\{ij\} \in best$ is taken to mean that edge $\{ij\}$ is traversed in that path, then the pheromone matrix M is updated at the end of each generation according to:

$$P_{ij}^{t+1} = \begin{cases} (1-\rho) \cdot P_{ij}^{t} + 1/f_{best} & \text{, if } \{ij\} \in best, \\ (1-\rho) \cdot P_{ij}^{t} & \text{, otherwise.} \end{cases} \tag{2}$$

The key factors which distinguish the *MAX-MIN* ACO are that the pheromone matrix P is initialised to its maximum value, is only updated with the information from the best ant per generation, and that the pheromone levels are truncated to a pre-specified range to avoid over saturation.

We would like to gratefully thank the authors and maintainers for the public provision of the ACOTSP package version 1.02 [24]. The original code was modified to call the same code as used by the EA for the fitness function and the new Local Search algorithm described in the next section. Substantial preliminary experimentation revealed that the default recommended settings appeared to provide robust performance. These were $\alpha = 1$ (exponent of pheromone used to calculate next-node probabilities), and $\rho = 0.02$ (pheromone decay rate). In Section 3.6 we describe the use of an additional matrix of heuristic "distance information". Experimentation with a range of values showed that results obtained were fairly robust to changes in the value of β (i.e., did not significantly differ), and so hereafter we report results with either the default setting of 2.0, or a value of 0.0 to remove this heuristic information. Different numbers of ants were used as described in Section 4.4. One significant change made, which resulted in a reversal in the rankings compared to the results we published in [22]. This was that the number of elapsed iterations of generating a complete set of ants with no improvement before the restart mechanism was invoked was greatly increased to 5000 generations.

3.4. Local Search Algorithm

The single-step greedy local search algorithm in Fig. 1 was implemented. Using problem-specific information in terms of task constraints, the perturbation operator moves one element to a different class as long as it does not leave behind an invalid class. Taking account of instance-specific information, a method can only move to a "receiving" class that has an attribute that it uses and vice-versa. This last condition does not assure that the fitness of the neighbour is superior to that of the incumbent, but rules out evaluating some of the least fit neighbours - at least in terms of coupling. Together these restrictions greatly reduce the size of the neighbourhood that needs to be examined, and so make local search more efficient.

3.5. Problem Specific Heuristics: Modified Pheromone Update

As noted above, the permutation representation creates redundancy, since the order in which elements appear within a class is immaterial. The following modification to Eq. 2 was designed to remove this effect, by increasing the pheromone laid down between all pairs of members of the same class in the best solution, and reducing it for all out of class members. Using $class^*[i]$ to denote the class label of element i in the best solution of a given iteration:

$$P_{ij}^{t+1} = \begin{cases} (1-\rho)\cdot P_{ij}^{t} + 1/f_{best} & \text{, if } i,j \leq a+m \text{ and } class^*[i] = class^*[j], \\ (1-\rho)\cdot P_{ij}^{t} - 1/f_{best} & \text{, if } i,j \leq a+m \text{ and } class^*[i] \neq class^*[j], \\ (1-\rho)\cdot P_{ij}^{t} & \text{, otherwise.} \end{cases}$$

Not shown here for reasons of space, are results using only the positive reinforcement (i.e., omitting the middle line of the equation above) which demonstrated worse performance.

3.6. Instance Specific Heuristic Information: Distance matrices

When applying ACO to routing problems such as the TSP, the heuristic matrix H can naturally encode for features such as the distance between nodes by setting $H_{ij} = 1/distance(i,j)$, to increase the probability of ants selecting nearby unvisited nodes as per Eq. 1. For the class modelling, there is no natural correlate of distance. However, as noted in Section 2.3, spectral graph partitioning approaches make use of a 0/1 "Adjacency Matrix" U formed by the edge set.

```
BEGIN
  /* given a starting solution $i$ */
  Decode $i$ into a set of classes $C$;
  set $f_i = calculate\_fitness(i)$ ;
  set set of possible moves $S = \{\}$ ;

  /* Build up a list of possible moves  */
  /* Taking the problem constraints into account */
  /* and not increasing coupling */
  FOR EACH  class $c \in C$ DO
    FOR EACH element $j \in c$ DO
      IF ($j$ is not the only attribute or method in $c$ ) THEN
        FOR EACH class $k \in C, k \neq c$ DO
          IF $j$ is a method and uses an attribute in $k$
          OR $j$ is an attribute and is used by a method in $k$ THEN
          set $S = S \cup \{j, k\}$;
          FI
        OD
      FI
    OD
  OD

  /* Greedy search over the possible valid moves */
  Randomise Order of $S$;
  set $improved = FALSE$;
  WHILE ( $S \neq \{\}$ AND $improved = FALSE$ ) DO
    take next possible move from $S$;
    apply move to create new solution $j$;
    set $f_j = calculate\_fitness(j)$;
    IF ( $f(j) < f(i)$) THEN
      set $i = j$;
      set $improved = True$;
    FI
  OD
END
```

Figure 1: Pseudocode of the Local Search algorithm

For each task, based on the use cases in the documentation and the numbering scheme above, an $l \times l$ matrix U was constructed with:

$$U_{ij} = \begin{cases} 1 & i \leq a, a < j \leq a+m, \text{ if method } j \text{ uses attribute } i, \\ 1 & j \leq a, a < i \leq a+m, \text{ if method } i \text{ uses attribute } j, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, since U_{ij} is the number of one-step paths between elements i and j, a useful result is that U_{ij}^n is the the number of length-n paths between i and j. We therefore calculate matrices U^n for a range of values of n and from these define a Distance Matrix D with elements $D_{ij} = MIN(n)$ such that $U_{ij}^n > 0$. The heuristic information matrix H is thereafter defined as $H_{ij} = 1/D_{ij}$, for all non-zero D_{ij} and zero otherwise.

In terms of class modelling, this means that if method i uses attributes j and k, then $D_{ij} = D_{ik} = 1$ and $D_{jk} = 2$. The same is true if i is an attribute used by methods j and k. We examined path lengths up to a distance of $n = 10$ - no problems examined had elements further apart.

4. Methodology

4.1. Quality Metrics

Many different quality measures have been proposed in the literature, and it remains an open question whether a multi-objective approach should be applied. However, it should be noted that the aim of class modelling is not to attain an approximation of the Pareto front since the extrema are of no interest. For example, one non-dominated solution always exists that achieves zero coupling by putting all elements into a single class. However, this goes against the whole spirit of object-orientated design, and in practice software designers deprecate this "anti-pattern", as it tends to lead to low cohesion (amongst other problems)[3].

Several authors have pursued the concept of "design elegance" in this and other fields. We have recently proposed several metrics that directly reflect a sense of "elegance" in terms of a symmetrical distribution of attributes and methods in a design[17].

In previous papers [18, 21] we have examined which metrics actually correspond to designers' preferences, via extensive interactive experiments wherein users' subjective judgements were used to build surrogate models. The results of these experiments demonstrated that the judgements made by a spread of users correlated highly to those of a surrogate model comprising a simple linear regression of a few metrics. Moreover, when presented with the candidate solutions obtained by meta-heuristic search using this model, users reported high degrees of satisfaction. This suggests strongly that the designs created did make sense to the users, despite having been created without any semantic knowledge of the labels on various elements such as *name*, *address*, *booking* etc.

Based on the regression co-efficients identified, we consider a single cost function (to be minimised) composed of two equally weighted elements:

$$f_{comb} = 0.5 * (f_{cbo} + f_{nac}) \tag{3}$$

The first element is based on the Coupling Between Objects (CBO) measure. The cost is defined as the percentage of all uses that are "out of class":

$$f_{cbo} = 100 \cdot \frac{\sum_i \sum_{j,class(j)\neq class(i)} U_{ij}}{\sum_i \sum_{j}, U_{ij}}. \tag{4}$$

The second cost element is the Numbers Among Classes (NAC):

$$f_{nac} = \frac{100}{6} * \left(\frac{\sigma_m}{2} + \frac{\sigma_a}{2}\right) \tag{5}$$

where σ_m and σ_a denote the standard deviations across all classes of the numbers of methods and attributes per class, truncated to the range $[0, 6]$. The lower this value, the more symmetrical the appearance of attributes and methods among the classes in the design, hence it tends to counterbalance the effect of the CBO metric.

4.2. Performance Metrics

Given that ultimately we are concerned with the use of these search heuristics embedded in an interactive design tool, that we assume the use of a surrogate fitness measure, and that we do not know an "optimal" fitness for each instance, we compare different search algorithms according to their effectiveness and consistency in finding good solutions.

For reasons of space and clarity we typically present the results graphically in the form of plots of mean performance over each algorithm-instance. SPSS v20 was used to carry out a two-way Analysis of Variance with 1000-fold boostrapping to allow for the possibility that the results are not normally-distributed. The cost of best solution found in a run was the dependant variable and fixed factors instance and algorithm, followed by post-hoc testing for significant differences at the 95% level using Tukey's HSD test. This test groups the results into homogenous subsets, so that the results for two variants may only be assumed to be statistically significantly different if they do not co-occur within any subset.

4.3. Problem Instances

To aid comparison with other published works, we used the three software design problems detailed in [16] which span a range of size and complexity. The first (CBS) is a generalized abstraction of a Cinema Booking System, the second (GDP) is a university system for student records, and the third (SC) is based on an industrial case study for booking cruise holidays. Results for manually produced designs are reproduced in Table 1, along with statistics about the problem instances. Please note that we are considering the early stages of design before a framework has been adopted, hence the number of classes is far smaller than it would be at a later stage.

It would of course be preferable to use a wider set of benchmarks. As we noted in Section 2.2, previous papers on this task have typically only used a single problem, and there are no accepted benchmarks. Neither is the required documentation typically available for large open source projects. A more robust approach typically used within the optimisation community is the use of parameterised randomised test generators. However, to have value it is necessary to identify appropriate parameters that can give rise to a range of problems exhibiting different sources of difficulties for search methods, and that knowledge is lacking in the field. Therefore we create a number of different variants of our problems to facilitate the identification of those factors which consistently affect the quality of solutions attainable.

Table 1: Measures of problems and their manual designs

Name	Instance Features			Manual Solution			
	Attributes	Methods	Uses	Classes	f_{cbo}	f_{nac}	f_{comb}
CBS	16	15	39	5	15.4	13.7	14.6
GDP	43	12	121	5	29.7	43.2	36.5
SC	62	30	126	16	45.2	25.33	34.3

The representation chosen permits models with variable numbers of classes to be present, since "end-of-class" markers could be adjacent in a permutation. Initially, to facilitate easy comparisons with the results produced manually (which for the SC represented several hours work) we imposed the condition that evolved designs should contain the same number of classes as the manual ones. Candidate solutions for the CBS and GDP problems are therefore required to have 5 classes, and SC has 16 (denoted SC16 below). As described later, we also considered versions of the SC problem with different fixed numbers of classes to examine the effect of increasing complexity.

4.4. Algorithm Parameters

The parameters specific to the ACO and EA are listed above. We ran experiments with 25, 50 and 100 individuals. Each algorithm was run one hundred times on each problem instance, with each run allowed to make 100,000 calls to the evaluation function. The algorithms resulting from different combinations of the search components are denoted as follows:

- A prefix *m-* denotes that the local search operator was applied to each candidate solution once created.
- *EA* and *AC* denote the global search methods EAs and ACO.
- A suffix of $-R$ denotes that a repair function was used to repair invalid solutions by moving elements from the most populated class to under-populated classes as needed.
- A suffix of -*C* denotes that a mechanism was used for reducing the redundancy caused by the sequence within a class. This is either an integer based representation with uniform crossover for the EA or the modified pheromone update mechanism for the ACO.
- A suffix of -*b* indicates that heuristic information was used within the constructive phase of the ACO.

Hence, for example, the variant *m-EA* is an EA using the permutation representation with edge recombination and local search, and *AC-RCb* is an ACO using the use-distance heuristic during path construction, with the repair mechanism applied to solutions once they are constructed and the modified pheromone update mechanism.

5. Results

No differences in the overall patterns of behaviour were observed when comparing results side-by-side for the three different population sizes, although a not-unexpected increased variability meant that fewer differences were statistically significant with 25 members. Thus for clarity and brevity we hereafter report results with a population size of 100.

5.1. Effects of Constraints and Repair

We begin by comparing the effect of the constrains implicit within our problem domain - namely that each class should contain at least one method and attribute. The approach taken was that designs with invalid classes were awarded a nominal fitness of 887 (a value chosen at random that is substantially bigger than the highest cost of 100.0 for a valid solution). This enables us to simply analyse our results to determine how many runs never found valid solutions, as a function of problem and algorithm.

The use of an EA as the global search component of the algorithm almost always enables the location of valid solutions. When it does not (EA on SC15), we see that redundancy avoidance, and/or the use of the repair function ensure search locate valid solutions. However, the use of Local search does not appear to affect the number of runs on which valid solutions are located (82 out of 100 in both cases). Given that the local search employed will perform an exhaustive search of the one-swap neighbourhood if necessary, this suggests that regardless of representation (redundancy avoidance), large swathes of the search space are invalid - effectively featureless plateaus given the penalty function we apply.

When ACO is the global search component, a very different picture emerges. With below 9 classes all runs locate valid solutions, but above 11 classes none do without the use of the repair function. In the transition case (11 classes), starting from a baseline of 25 runs locating valid solutions for the ACO alone, and 12 for the memetic version m-AC, the following effects are notable:

- adding the distance information makes matters worse – the number of “successful” runs drops to 16(AC-b) and 6(m-AC-b);
- avoiding redundancy improves the situation – the number of successful runs increases to 42 (AC-C) and 50 (m-AC-C);

- there are synergies between the three additional components. The number of "successful" runs rises to 92 for AC-Cb. Recalling the form of Eq. 1, and noting that the effect on the modified pheromone update is that more components of P will have their rate of decay reduced, this might suggest that the effect of the distance-based heuristic information is too great. On the other hand, the cost of a "tour" as measured by the combined cost function is between 0 and 100, whereas the lower bound on the estimated cost of a nearest-neighbour tour generated by our distance matrix would be N, so clearly some kind of problem-specific scaling mechanism may be necessary.

Given these results, for the sake of improving the clarity of the figures and tables, hereafter we only report results using the repair function.

5.2. Effect of Search Components on Quality of Solution

Figure 2 shows the mean (top) and absolute (bottom) values of the best solutions found, with algorithms on the x-axis and a separate line for each problem. As can be seen, the use of an EA as the global search component gives better performance across all problem types. Note that in the top figure the ordering of algorithms on the x-axis, with local search alternately absent/present creates a sawtooth shape of lines for each problem, illustrating the benefits of local search. Note also that for the SC problem with fewer than 11 classes (dotted/dashed lines) the curves are fairly flat, whereas the effect of different search components is more noticeable for more complex problems (SC11 and above - solid lines). In the bottom figure the sawtooth effect for the ACO values now starts at SC10 rather than SC11. The differences in the values of the best values ever found (i.e., allowing for multiple runs) is smaller between EA and AC, and in fact for the CBS problem the best solutions ever found are discovered by AC-R, m-AC-R,AC-Rb and m-AC-Rb.

Table 2 shows the statistically significantly different sub-groups with their mean values for all problems. The following observations can be made:

- The use of an EA as the global search component of search means runs of the search process discover lower cost solutions than using ACO.
- In every case except EA-RC (where the difference is not significant), the addition of local search enables discovery of lower cost solutions.
- Redundancy avoidance helps the EA.
- Neither redundancy avoidance nor distance information aids the ACO.

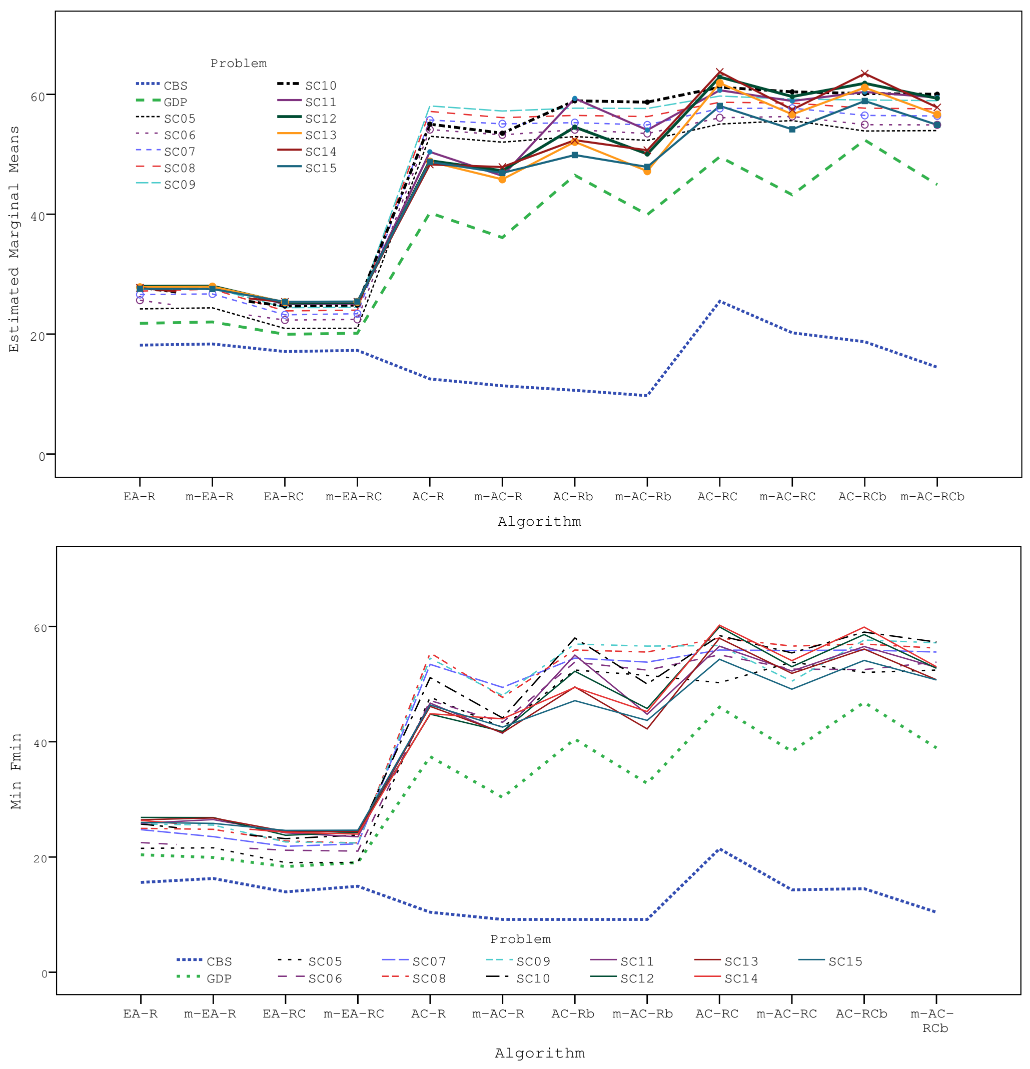

Figure 2: Overall (bottom) and Mean (top) of best solution found for each problem.

5.3. Effect of Problem Characteristics

To better understand the influence of characteristics of the search problem, a local search algorithm was run 100,000 times on each problem using both permutation and integer (i.e., redundancy avoiding) encodings. Each run started with a randomly created candidate solution, and applied the 1-move local search method above until no further improvements could be

Table 2: Results of testing to discriminate homogenous subsets. Values in cells are means for lowest cost found per run.

	Subset									
Algorithm	1	2	3	4	5	6	7	8	9	10
EA-RC	23.3									
m-EA-RC	23.4									
m-EA-R		25.8								
EA-R			26.0							
m-AC-R				46.8						
AC-R					48.5					
m-AC-Rb					48.7					
AC-Rb						50.8				
m-AC-RCb							53.0			
m-AC-RC								53.7		
AC-RCb									55.3	
AC-RC										56.2

found. For each of these local optima was stored its cost, the difference in cost, and the number of different values (i.e., the extension of Hamming Distance) from the lowest cost solution found during the 100,000 runs.

Table 3 displays the lowest cost solutions found, with the hand-crafted results for comparison. Immediately noticeable is that despite the variation in mean search results noted above, the best solution found for each problem does not change to anything like the same extent. To put this another way, it would appear that neither the scale nor complexity of the problem greatly affect the cost of the global optimum, but they do make the landscape harder to search. This is equally true for both encodings, which of course have different genotype-phenotype mappings, and hence present different landscapes to the global search element. We note that in every case meta-heuristic search was able to find solutions better than the provided hand-crafted solutions, and inspection showed that these made semantic sense.

The first characteristic investigated was the number, quality and distribution of local optima within the search landscapes. Given the size of the search space, and that graph-partitioning is NP-complete, the lowest cost ever found was used as an estimate of the global optimum cost f^*_{comb} for each

Table 3: Lowest Cost solutions found by different search algorithms for each problem.

Enco ding	Search Method	CBS	GDP	SC5	SC7	SC9	SC11	SC13	SC15
Perm.	mAC-R	9.2	30.3	42.5	49.4	48.0	41.5	41.5	42.5
	mEA-R	16.3	19.9	21.5	24.7	25.7	25.9	26.5	25.8
	LS	16.9	20.4	21.6	26.5	25.7	28.1	27.8	27.2
Int.	mEA-RC	14.9	18.3	19.0	21.9	22.4	23.5	24.4	24. 6
	LS	16.3	19.8	20.9	22.7	24.1	24.7	25.1	25.2
Human		14.6	36.5						34.3
Best		9.2	18.3	19.0	20.5	22.4	20.6	20.6	21.6

problem. The top pane of Figure 3 shows for each problem the distribution of the normalised fitness differences - that is to say, of $(f_{comb} - f^*_{comb})/f^*_{comb}$. As can be seen, the local optima in the permutation-based landscapes have higher costs (larger normalised differences) than their counterparts in the integer-encoded redundancy-avoiding landscapes. In both cases there is a very wide spread of values, and given that the median normalised difference is mostly over 0.5, most of the local optima have costs more than 150% of the global best. Apart from the CBS results, it is also noticeable that the distributions of values are fairly constant for the integer encodings. However for the permutation encodings they rise from GDP through to SC5 and SC7 (where most local optima costs are twice the global best) before reducing as the number of classes increases. These findings are line with the results of the algorithm comparison above, especially for ACO, and suggest a reason for those results - that ACO is getting stuck in local optima, and that these tend to have higher fitness values for some problems.

The bottom pane of Figure 3 shows the normalised distance - that is, proportion of elements with a different class label to that in the estimated global optimum. This shows that typically local optima have no elements with values in common to the estimated global optimum for permutation encodings. For the redundancy-avoiding integer encoding, it is noticeable that CBS,GDP and SC5, which all have 5 classes, have similar distributions, and that the typical distance rises thereafter for the SC problems.

Figure 4 brings these results together in scatter plots of normalised fitness

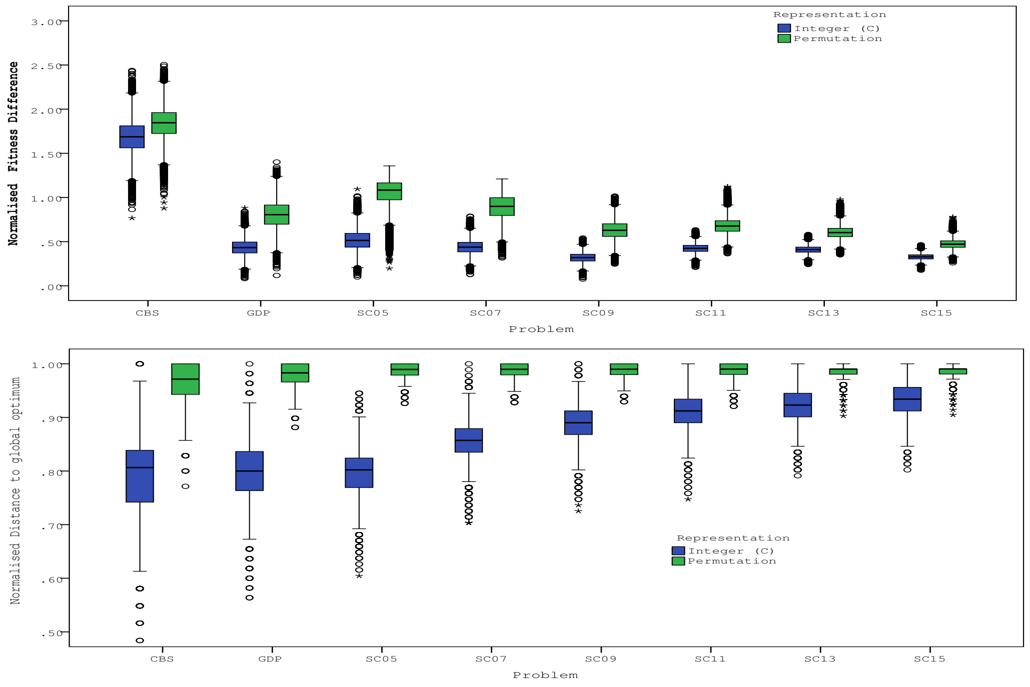

Figure 3: Box plots of normalised fitness difference between (top), and distance to (bottom) local and global optimum. Boxes indicate interquartile range, line is median. Circles denote outliers more than 1.5 and 3 box-widths from quartiles and asterisks more than 3.

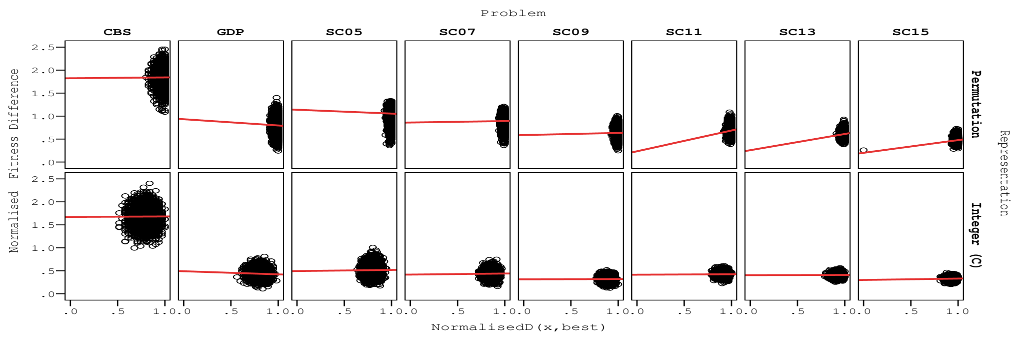

Figure 4: Scatter plots of local optima relative fitness vs. distance to optima for each problem and representation. Lines represent the least-squares fit.

difference vs normalised distance, with regression lines fitted. Although the co-efficient of correlation is fairly low, it is worth noting the slope of the lines. These provide clues to the reason for worse performance observed for ACO and local search on GDP, SC5 and SC7 on the permutation landscapes: there is a negative correlation between cost difference and distance. In other words the more highly fit local optima tend to be further from the global optimum. In contrast, for the SC11,SC13 and SC15 problems the correlation is positive, and so moving from one local optimum to a better one will, on average, lead towards the global optimum.

To examine how much of this was the effect of the redundancy in the encoding, we re-ran the local search experiments for the 5-class problems with integer encoding. At the end of each experiment we generated all $5! = 120$ possible versions of the best solution by permutating the class labels. Then for each local optima we recorded the distance to the closest copy of the global optimum. Again, measuring the excess cost, and normalised distance, we examined the spread of these measures as shown in Figure 5. As can be seen from the top two panes, the distributions of distances and excess cost follow a normal distribution in all three problems. Comparing excess cost to the pooled mean and median values, the CBS distribution contains more high cost solutions and the GDP fewer. Comparing the distances, it is noticeable that the spread of values is far wider, and the mean of the distribution lower, for the CBS than the GDP, which in turn is wider than the SC. This is clear from the scatter plot as well. Since there are more local optima closer to the global optimum for CBS, it follows that there is a higher chance of escaping from an arbitrary local optimum (e.g., reached early in randomly initialised search) to a global optimum on that problem.

The next aspect to be considered is the effects of scale and complexity. Table 4 shows the co-efficient of determination R^2, a measure of the amount of the observed variation explained by a linear regression model of the cost of the solutions found in terms of different problem descriptors. The rows give the results from the 100,000 iterations of local search on two different representations, and from the meta-heuristic search algorithms (100 runs per algorithm variant). Scatter plots showed that in every case the correlation was weakly positive - the cost of the local optima found increased in line with the different measures of problem scale or complexity.

- For all but the EA with integer representations (EA-RC/m-EA-RC) the association with the number of classes is extremely weak.

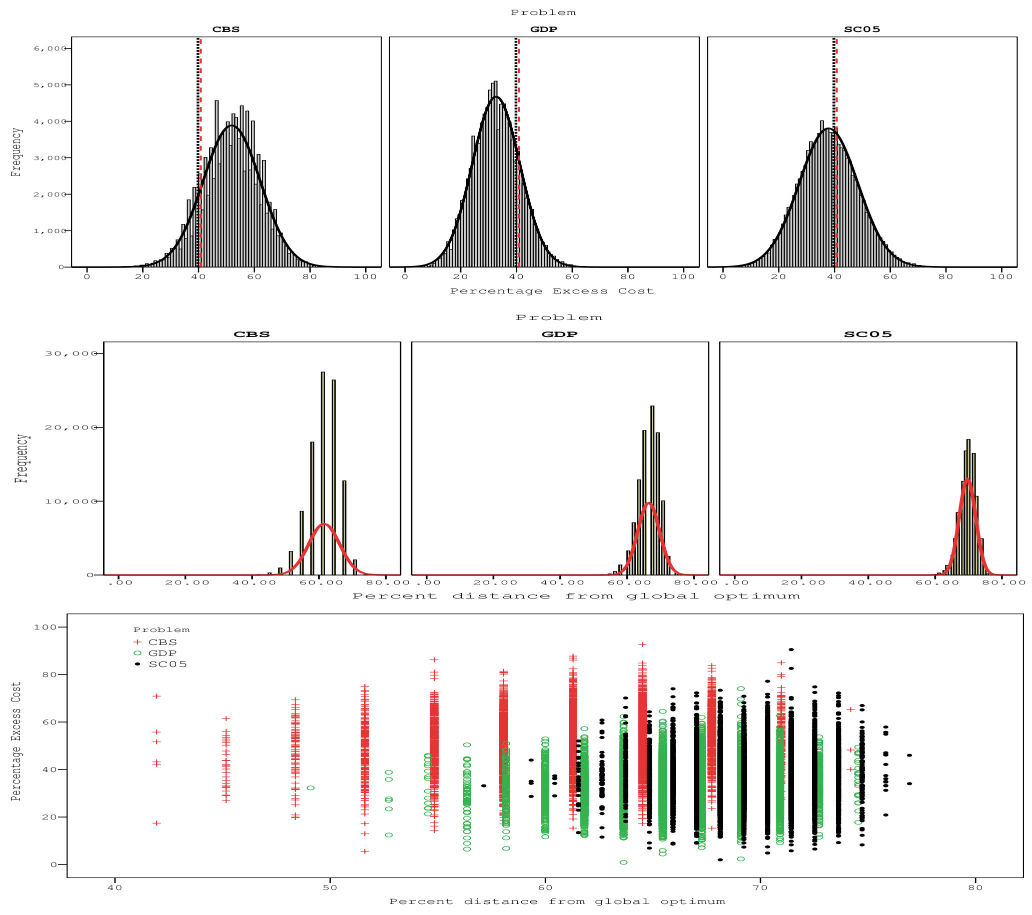

Figure 5: Histograms of excess cost (Top), and distance to closest copy of global optimum (Middle), and Scatter plots (Bottom) showing the distribution of local optima for the five class problems. Reference lines show the mean and median of the pooled distributions

- For all algorithms the number of uses per method is insignificant.
- All other effects appear significant, but since they only have three values they are effectively serving as proxies for the problem instance.
- The degree of correlation is insignificant ($R^2 < 0.15$) for the other possible predictors "uses/methods/attributes per class".

To delve further into these results we used the automatic linear regression tool in SPSS to build a model for predicting the lowest cost per run. This

Table 4: Co-efficients of determination for linear models relating various measures of problem characteristics to the observed quality of solutions.

R^2	Best vs. Classes	Attributes	Methods	Uses	Uses per method	Uses per attribute
LS int.	0.155	0.529	0.487	0.398	0.025	0.378
LS perm.	0.005	0.477	0.294	0.461	0.006	0.175
EA int.	0.673	0.692	0.615	0.547	0.031	0.471
EA perm	0.489	0.816	0.723	0.644	0.003	0.559
AC perm.	0.148	0.836	0.544	0.805	0.034	0.358

procedure converts continuous variables into ordinal ones, then uses forward stepwise predictor selection based on information criteria to build a model based on this series of binary decisions (e.g.,"classes= 5", "classes= 6", etc.). Therefore the choice of effects selected, and their relative importance, gives insight into the factors that affect the quality of results found.

For the results of EA-RC/m-EA-RC the model produced yielded a predictive accuracy of 96.6%. The relative importance of classes was 0.746, that of the transformed variable "attributes=16" was 0.254, all other effects were removed by modelling process as insignificant.

By way of contrast, repeating this process for the pooled results with the permutation representation only yielded a model with a 44.5% predictive accuracy. The importance of the transformed value of Uses per Attribute was 0.915, that of classes was 0.085, all others were removed. If the tool was allowed to use the type of global search component, the predictive accuracy increased to 88.2%. The relative importance of effects selected in the model was global search (0.701), attributes 0.278 and classes 0.014.

6. Analysis

To a large extent the results of the statistical landscape analyses merely confirms what the figures above have already suggested: namely that the principal factor determining the quality of solution found appears to be the choice of search heuristic. For all of the problems investigated it appears that solutions exist with similarly low cost f_{comb}, the difficulty lies in locating these in the presence of many local optima. To give an idea of the number of these, there were less than ten duplicates in the 100,000 local optima located per problem. Moreover, the local optima appear to become both higher cost, and on average further from the global optimum as the complexity of the problem is increased.

Taken together with the results for search performance, Figure 5 shows a clear relationship between the distribution of local optima within the search space, and the ability of search algorithms to reliably locate solutions close to the estimated global minimum cost.

For all but the simplest problem (CBS), EA-based variants outperformed their ACO-based counterparts, and behaved robustly with respect to problem scale and complexity. The two ways of adding of problem-specific knowledge to the EA (reducing redundancy via the choice of encoding, and using a repair function) were both beneficial, as was the use of instance-specific information in the local-search algorithm.

In contrast, although the ACO required the repair function for the more complex functions, adding information via distance heuristic, and reducing redundancy were both detrimental to the ACO-based variants. We hypothesize that both of these may have the effect of focussing search, causing premature convergence. To test this, we examined ACO runtime logs, which revealed that the lowest cost solutions are typically discovered an order of magnitude sooner than the equivalent EA-based experiments. This suggests that basins of attraction of fewer local optima are sampled during search. Given the relationship between the quality of local optima and their distance from the global optima (and hence from other low cost solutions) shown in Fig. 4 this explains the performance curves seen in Fig. 2. For some mid-scale problems (SC5-7) the negative quality-distance correlation means that algorithms which move from local optima to local optima will actually move away from the global optima, and because of the redundancy of the permutation encoding will actually be moving into more sparsely populated regions, where they are more likely to to become trapped. There remains the possibility that our results would dramatically change if we could find some "magic" set of settings that reduced this problem of ACO premature convergence, since we did not exhaustively tune the parameter values. However, our preliminary investigations did include all combinations of several different values for each parameter, in addition to the recommended settings.

The principal threats to the validity of this analysis are two-fold: that we have only used three base problems, and that the artefacts may arise from our choice of fitness function.

It should be noted at the outset that the field suffers from a lack of benchmarks. Our previous papers use three problems, the other papers cited in Section 2.2 use only one [2, 27]; or two [15]. Clearly this poses a threat to the validity of those results, and our findings above. The development of a reliable benchmark set would greatly aid comparison of experimental methods. Elsewhere we have reported on the design of tuneable randomised test landscape generators where various factors such as the problem size, the number and relative size of local optima, the degree of epistatic interference between partitions of the search space, and "deceptiveness" could be tuned to facilitate algorithm design and analysis [23]. Hence some of our experimentation into the factors that appear to make problems hard for SBSE. It would appear that, although the quality of the *best* solution present does not change much for our weighted-sum metric, the problem size (effectively the size of the graph to be partitioned), and the number of uses per attribute (closely related to the degree of the graph) are the major factors in determining the reliability with which the lowest cost solutions can be found.

The question of the cost metric used is more subtle. In earlier papers we reported results from f_{CBO} on its own, as well as in combination with f_{NAC}, which were broadly in line with our results here. It remains for future work to consider the effects of incorporating other measures of elegance or of cohesion etc., and of course it could be argued that it might be more appropriate to adopt a multi-objective approach, modifying a MOEA to focus away from the "ends" of the pareto front. However, as we have pointed out before, the proliferation of proposed quantitative metrics outstrips the current limits of MOEA algorithms, and moreover we have evidence that human qualitative judgements correspond closely to surrogate models built from a few elements.

As well as given an indication of the value of different search components in the face of increasing problem instance scale and complexity, the results also provide possible insights into what makes the class modelling problem hard for software engineers to perform. Firstly, software engineers typically do not exploit use information (available in use case descriptions or user stories) when manually evaluating their candidate class models. However, the above results show that it is beneficial to exploit this information, especially as heuristic knowledge appears to be generally beneficial to search.

7. Conclusions

We preface our remarks by the acknowledging that the lack of benchmarks, especially large-scale instances for this task mean that we must be somewhat cautious in our findings. Nevertheless, our results have gone some way towards identifying the problem characteristics that are important to vary when selecting or constructing a range of test instances.

For early lifecycle design of object-oriented class models, and given the computational budget allowed, using Evolutionary Algorithms as the global search component of an algorithm outperforms the use of Ant Colony Optimisation. EAs are more capable at handling constraints, and the influence of different search components such as repair functions, redundancy avoidance and local search is beneficial to the search process. Although competent on the less complex problems, using ACO on the more complex solutions often failed to find valid solutions without the use of a repair function. Other methods for incorporating problem-specific information appeared to exacerbate the tendency of the ACO to prematurely converge. On certain problems this causes a rapid decrease in performance, and landscape analysis revealed that this coincided with a landscape structure exhibiting a positive fitness-distance correlation, where low-cost local optima are more typically further from the global optimum than higher cost ones.

To better understand the search landscapes, and how these related to problem characteristics, we used iterated random-restart local search to probe the local optimum structure of the landscape. All problems contained a huge number of local optima, with little structure to their distribution relative to the estimated global optimum. Both increasing complexity (classes) and scale (number of problem elements to be grouped) caused the cost of local optima to increase. Interestingly, however, there was not a corresponding change in the cost of the best solution ever found, which we use as a proxy for the global optimum.

In other words as problems become harder and more complex, equally good solutions are still out there to be found, the challenge for search algorithms lies in finding them amongst a plethora of local optima, often leading in misleading directions. In future work we aim to deepen our understanding of this effect, address the problem of a lack of benchmarks in the field, and apply this analysis of the effects of search components and problem characteristics to other branches of SBSE.

References

[1] C-E. Bichot and P. Siarry. *Graph Partitioning: Optimisation and Applications.* ISTE Wiley. 2011.

[2] M. Bowman, L. C. Briand, and Y. Labiche. Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms. *IEEE Transactions on Software Engineering*, 36(6):817–837, Nov. 2010.

[3] W. Brown, R. Malveau, H. McCormick, and T. Mowbray. *Anti-Patterns: Refactoring Software, Architectures, and Projects in Crisis.* Wiley, 1998.

[4] B. Curtis, H. Krasner, and N. Iscoe. A Field Study of the Software Design Process for Large Teams. *Communications of the ACM*, 31(11), 1268-1287. 1998.

[5] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer: Heidelberg, Berlin, New York, 2003.

[6] E. Falkenauer. *Genetic Algorithms and Grouping Problems* Wiley. 1998.

[7] R.L. Glass. *Facts and Fallacies of Software Engineering.* Addison-Wesley. 2003.

[8] Graph Partitioning Archive. http://staffweb.cms.gre.ac.uk/wc06/partition/ accessed November 2013.

[9] N. Krasnogor and J. Smith. Competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9:474–488, 2005.

[10] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd Ed.* Dorling Kindersley Pvt Ltd. 2008.

[11] R. Lewis and E. Pullin. Revisiting the Restricted Growth Function Genetic Algorithm for Grouping Problems *Evolutionary Computation* 11:3 693–704. 2011.

[12] S. Meyer-Nieberg and H.-G. Beyer. Self-adaptation in evolutionary algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pp. 47–75. Springer, 2007.

[13] M. Petre. Insights from Expert Software Design Practice. In *Proc. Joint 12th European Software Engineering Conf. and 17th ACM SIGSOFT Symp. on the Foundations of Software Engineering*, pp 233-241. 2009.

[14] M. Serpell and J. Smith. Self-Adaption of Mutation Operator and Probability for Permutation Representations in Genetic Algorithms. *Evolutionary Computation*, 18(3):1–24, Feb. 2010.

[15] O. Sievi-Korte, E. Mäkinen, and T. Poranen. Simulated Annealing for Aiding Genetic Algorithm in Software Architecture Synthesis. *Acta Cybernetica* 21:2,235-265, 2013.

[16] C. Simons. Case study specifications, available at http://www.cems.uwe.ac.uk/~clsimons/casestudies.

[17] C. Simons and I. Parmee. Elegant object-oriented software design via interactive, evolutionary computation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 42(6):1797–1805, 2012.

[18] C. Simons, I. Parmee, and R. Gwynllyw. Interactive, Evolutionary Search in Upstream Object-Oriented Class Design. *IEEE Transactions on Software Engineering*, 36(6):798–816, 2010.

[19] C. Simons and J. Smith. A Comparison of Evolutionary Algorithms and Ant Colony Optimization for Interactive Software Design. In *Proceedings of the 4th Symposium on Search Based-Software Engineering*, p.37, 2012.

[20] C. L. Simons and J. E. Smith. A Comparison of Meta-heuristic Search for Interactive Software Design. *Soft Computing* 17(11):2147–2162, 2013.

[21] C.L. Simons, J. Smith and P. White. Interactive ant colony optimization (iACO) for early lifecycle software design. *Swarm Intelligence*, 8(2):139-157, 2014.

[22] J.E. Smith and C.L. Simons. A Comparison of Two Memetic Algorithms for Software Class Modelling. In *Proceedings of the 15th Annual Genetic and Evolutionary Computation Conference*, pp. 1485 - 1492, ACM, 2013.

[23] R.E. Smith and J.E. Smith New Methods for Tuneable, Random Landscapes. *Foundations of Genetic Algorithms*, pp. 47–67. Morgan Kaufmann. 2001.

[24] T. Stuetzle. ACOTSP, version 1.2, available at http://www.aco-metaheuristic.org/aco-code.

[25] T. Stuetzle and H. H. Hoos. Max-min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[26] D. Svetinovic, D.M. Berry, and M. Godfrey. Concept identification in object-oriented domain analysis: why some students just don't get it. In *Proceedings of the International Conference on Requirements Engineering*, 189-198. IEEE Computer Society, 2005.

[27] S. Vathsavayi, H. Hadaytullah, and K. Koskimies. Interleaving human and search-based software architecture design. *Proceedings of the Estonian Academy of Sciences*, 62(1):16, 2013.

[28] G. Wang. *Ant Colony Metaheuristics for Fundamental Architectural Design Problems.* PhD Thesis, Uni. California in Santa Barbara, 2007.

[29] R. Wirfs-Brock, and A. McKean. *Object Design: Roles, Responsibilities, and Collaborations.* Addison-Wesley. 2003.

[30] Y. Zhang. Repository of publications on search-based software engineering. Available at http://crestweb.cs.ucl.ac.uk/resources/sbse_repository.