Soft Comput (2013) 17:2147–2162
DOI 10.1007/s00500-013-1039-1

METHODOLOGIES AND APPLICATION

A comparison of meta-heuristic search for interactive software design

C. L. Simons · J. E. Smith

Published online: 3 April 2013

Abstract Advances in processing capacity, coupled with the desire to tackle problems where a human subjective judgment plays an important role in determining the value of a proposed solution, has led to a dramatic rise in the number of applications of Interactive Artificial Intelligence. Of particular note is the coupling of meta-heuristic search engines with user-provided evaluation and rating of solutions, usually in the form of Interactive Evolutionary Algorithms (IEAs). These have a well-documented history of successes, but arguably the preponderance of IEAs stems from this history, rather than as a conscious design choice of meta-heuristic based on the characteristics of the problem at hand. This paper sets out to examine the basis for that assumption, taking as a case study the domain of interactive software design. We consider a range of factors that should affect the design choice including ease of use, scalability, and of course, performance, i.e. that ability to generate good solutions within the limited number of evaluations available in interactive work before humans lose focus. We then evaluate three methods, namely greedy local search, an evolutionary algorithm and ant colony optimization (ACO), with a variety of representations for candidate solutions. Results show that after suitable parameter tuning, ACO is highly effective within interactive search and out-performs evolutionary algorithms with respect to increasing numbers of attributes and methods in the software design problem. However, when larger numbers of classes are present in the software design, an evolutionary algorithm using a naïve grouping integer-based representation appears more scalable.

Keywords Interactive search · Meta-heuristics · Software design · Search-based software engineering

Communicated by G. Acampora.

C. L. Simons (✉) · J. E. Smith
Department of Computer Science and Creative Technologies, University of the West of England, Bristol BS16 1QY, UK
e-mail: chris.simons@uwe.ac.uk

J. E. Smith
e-mail: james.smith@uwe.ac.uk

1 Introduction

The application of automated search to a range of software development activities has attracted significant research attention. Indeed, Search-Based Software Engineering (SBSE) (Harman 2007, 2011) is now a well-established discipline. SBSE historically focused on software testing where solutions can be represented fairly naturally and metrics such as structural and functional test coverage can be automatically calculated to serve as quality functions. However, in the upstream stages of the software design, such as the object-oriented modeling of design classes, the choice of evaluation functions is much less well defined. To give one example, Bowman et al. (2010) cite 6 different possible metrics relating to the structural integrity of the design with respect to design coupling and cohesion. Here the precise balance of factors affecting the subjective judgments of the human software engineer is less well understood—hence the oft-heard references to the "art" of software design. Indeed, this is precisely the sort of scenario in which Interactive Evolutionary Algorithms (IEAs) have been shown to perform well [see e.g. the survey in Takagi (2001), and more recent work such as Takagi and Ohsaki (2007); Caleb-Solly and Smith (2007); Brintrup et al. (2008); Pauplin et al. (2010)]. Our earlier work demonstrates that we can indeed successfully use meta-heuristics to provide computational support for an interactive software design process, evolving object-oriented class models that met designers' criteria—both subjective (Simons et al. 2010) and aesthetic (Simons and Parmee 2012).

As with most papers in the field, such interactive design search uses an Evolutionary Algorithm (EA) (Eiben and Smith 2003) because of their long history of successful applications. However, as the name of the field of SBSE suggests, potentially any search algorithm could be used, although in practice research effort has also tended to concentrate on meta-heuristics, in particular EAs. It is appropriate that we challenge adoption of a technology based on history, and examine whether other search methods might be better suited to some, if not all, interactive design search tasks. Indeed the same argument has been made for SBSE in general: "We must be wary of the unquestioning adoption of evolutionary algorithms merely because they are popular and widely applicable or because, historically, other researchers have adopted them for SBSE problems; none of these are scientific motivations for adoption" (Harman 2011).

One major contribution of this paper is to identify a number of factors that we believe are crucial to making an informed choice for an underlying search engine for interactive search (Sect. 4). Then, to make the comparison concrete, we describe the experimental methodology followed and three case studies of early lifecycle software design tasks (Sect. 5). Results of comparing the different algorithms according to the factors identified are presented in Sect. 6, and finally in Sect. 7, we conclude by making some recommendations for possible users of interactive search tools.

2 Background

2.1 Search-based software engineering

Search-based software engineering is an approach that applies meta-heuristic search techniques such as simulated annealing, tabu search and genetic algorithms to address software engineering (SE) problems. It is motivated by the observation that many aspects of the SE process can be formulated as optimization problems, and as such are amenable to automated search. However, due to the size and computational complexity of such SE problems, exact optimization techniques such as linear programming or dynamic programming are impractical. In most cases the search suffers from combinatorial explosion, and the "fitness" landscapes are thought to exhibit discontinuities and multiple optima. Because of this, researchers and practitioners alike have widely used meta-heuristic search techniques such as EAs not only to gain insight into SE problems but also to arrive at near optimal or 'good-enough' software solutions.

Early attempts to apply optimization to SE problems harnessed EAs to evolve software sequences of test cases based on executable program branch coverage as fitness functions (e.g. Xanthakis et al. 1992; Smith and Fogarty 1996; Jones et al. 1996) and microprocessor design verification tests (Smith et al. 1997). Later, the term "Search Based Software Engineering" (SBSE) was coined around the turn of the millennium by Harman and Jones (2001). In the last decade, the number of applications of SBSE has increased greatly and reports can be found across the spectrum of the SE lifecycle. Examples include requirements analysis and scheduling (Ren et al. 2011), design tools and techniques (Bowman et al. 2010; Simons et al. 2010), software testing (McMinn 2004), automated bug fixing (Weimer et al. 2010), and software maintenance (O'Keeffe and Cinneide 2008). Harman (2011) provides a useful overview, while a comprehensive repository of SBSE publications is maintained by Zhang (2012).

2.2 Object-oriented software design

The first stage in software design is to identify and evaluate the concepts, rules and information relevant to the design problem domain under investigation. Using the object-oriented paradigm, these elements of the design problem domain are represented using the 'class' construct, wherein individual instances of classes are known as objects. These classes and objects are the key 'building blocks' of a software system and so have crucial relevance to subsequent downstream software implementation and testing. Various manual approaches to the identification and evaluation of appropriate classes have been proposed, such as the use of abstraction for classical categorization (Booch 1994) and class responsibility assignment (Wirfs-Brock and McKean 2003). However, this remains a cognitively demanding and challenging task for the software engineer to perform, not least because of the many competing requirements of a design problem.

The Unified Modeling Language (UML) (Booch et al. 1999; Object Management Group 2012) is the standard modeling language of the object-oriented paradigm, and is widely used by software designers. According to Booch et al. "the UML is a graphical language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system". Using the UML, classes are placeholders or groupings of attributes (i.e. data that need to be stored, computed and accessed), and methods (i.e. units of execution by which objects communicate with other objects, human users, or programs, etc.). Each class, attribute and method is a discrete model element. Thus early lifecycle software design involves identifying attributes and methods from the design problem, and then finding an appropriate partitioning of these attributes and methods into classes. We will henceforth refer collectively to methods and attributes as "elements", and candidate partitions into classes as "designs". To ensure that each class is meaningful as a relevant concept to human designers, we impose the constraint that each class contains at least one attribute and at least one method. A small class diagram showing UML notation of a design with three classes is shown in Fig. 1. Each class thus denotes a concept or abstraction relevant to the design problem domain. Where appropriate, there exist relationships (or couples) between classes where one class depends upon another to fulfill its capabilities; this is denoted by solid arrows with an open arrow-head indicating the direction of couple. It is important to note that there is no ordering among the partitions (classes) of the notation.

2.3 Formulating class modeling as a search problem

In formulating software design as a search problem, the class construct can be represented in a number of ways (see Sect. 3). Common to all of these is that to enable efficient search, it is necessary to assign a quality measure to evaluate candidate solutions. A large number of quantitative metrics have been identified, many referring in different ways to the structural integrity of a design with respect to "coupling" (the extent to which one class depends on others to fulfill its capabilities) or "cohesion" (the extent to which a class has clear purpose) e.g. Harrison et al. (1998), Briand et al. (1999), Al Dallal and Briand (2010). It is generally held by software engineers that a good software design should exhibit low coupling and high cohesion, but these are potentially competing measures. Indeed, this is a typical example of how many design decisions can be balancing acts, trading-off one quality measure against another. This has led some authors to use multi-objective, quantitative approaches to search e.g. Bowman et al. (2010), Harman and Tratt (2007). However, our work has taken a different approach: rather than attempt to define coupling/cohesion metrics which capture what a user is looking for, and then explicitly manage the quantitative multi-objective trade-off, we have used a multi-objective IEA where the designer is responsible for assigning qualitative fitness to a candidate solution (Simons et al. 2010). To relieve the burden of interaction fatigue on users, we have also investigated the use of a surrogate fitness function that learns a model of qualitative "elegance" from the users' decisions (Simons 2011; Simons and Parmee 2012), so that not all solutions need be manually evaluated.

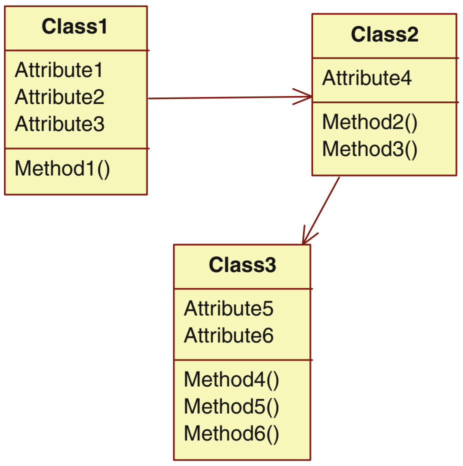

Fig. 1 Small example class design in UML notation

2.4 Interactive meta-heuristic search

Interactive EAs were popularized in Dawkins' 'biomorphs' program (1990), but build on a well-established field in Artificial Intelligence. They have been successfully applied in a wide range of applications to facilitate user-personalization without the need for time consuming explicit knowledge-acquisition process (Takagi 2001). Typically the user is presented with a number of solutions, and rates them according to the extent to which they match the user's desiderata. Thus this process implicitly captures the user's multi-objective decision making processes. Well known early applications include face-recognition (Caldwell and Johnston 1991), the evolution of computer graphics (Sims 1991a, b), and hearing aids (Ohsaki et al. 1998). High-profile recent successes include tuning Cochlear Implants (Legrand et al. 2007).

A common feature of IEAs is their reliance on human guidance and judgment to direct and control the search, which creates both potential weaknesses and strengths. On one hand, human assessment tends to have a component of subjectivity and non-linearity of focus over time. Thus including a human in-the-loop introduces a need for rapid convergence to prevent the interactive process from becoming tedious for the human participant. At the same time the ability to maneuver the search interactively can potentially be exploited as a powerful strategy for adapting an otherwise naive EA.

There have been a number of studies addressing the issues related to minimizing fatigue both, physical and psychological, that can result from prolonged interaction times and the possible stress of the evaluation process. Discretizing continuous values to using five or seven levels was shown to facilitate decision making when allocating fitness values, without the quantization noise significantly compromising convergence (Ohsaki et al. 1998), and this limit on capacity for processing information has been comprehensively discussed in Miller (1956) where he suggests organizing the information into several dimensions and successively into a sequence of "chunks" could help stretch this limit on bandwidth.

Alternative ways of reducing time taken to discover good solutions is by considering larger population sizes and using a screening mechanism whereby only a few individuals showing good fitness are displayed to the user. Several methods have been proposed to approximate fitness by, for example, clustering individuals (Lee and Cho 1999; Boudjeloud and Poulet 2005) or using multiple fuzzy state-value functions to approximate the trajectory of human scoring (Kubota et al. 2006). An interactive concept-based search using a multi-objective evolutionary algorithm was proposed by Avigad et al. (2005) which combined a model-based fitness of sub-concept solutions (using a sorting and ranking procedure) with human evaluation. The efficacy of combining qualitative (user-provided) and quantitative (computer-generated) objectives was also demonstrated in Brintrup et al. (2008). Within SBSE, design elegance has been exploited as a model and surrogate fitness function that then is dynamically combined with quantitative objectives to produce elegant software designs (Simons and Parmee 2012).

Surprisingly, the literature does not seem to contain many examples of the use of alternatives to EAs as the underlying heuristic for interactive search. Rather, approaches rely on using either a meta-heuristic with a defined quality function and periodically using user interaction to guide search by reformulating a fitness function or preference weighting, or simply to change the search characteristics via changes to the algorithm parameters. Examples of the former include multi-objective Iterated Local Search (Geiger 2008) and Tabu Search (Kopfer and Schonberger 2002) and of the latter include Ant Colony Optimization (ACO) (Uğur and Aydin 2009). Interestingly, however, there is one report of interactive search with Particle Swarm Optimization used to design temperature profiles for a batch beer fermenter (Madar et al. 2005).

In a further interactive approach, a user-centric memetic algorithm has been proposed by Badillo et al. (2013) who report that interactive memetic algorithms are capable of taking advantage of good quality human feedback, not merely as a carrier of subjective information, but also as a source of problem-aware perturbations that can drive/focus the algorithm to specific regions of the search space. In other words, an interactive memetic algorithm can behave in a proactive manner i.e. it attempts to anticipate the user's behavior and then exhibit some degree of creativity. Memetic algorithms have also been applied to adaptive agent-based machine learning (Acampora et al. 2011) and for solving the ontology alignment problem (Acampora et al. 2012).

3 Choice of meta-heuristic search algorithms

There are of course many meta-heuristics in the literature, and it would not be feasible to compare all within the space of a paper. Our selection of meta-heuristics for consideration has been driven by the following considerations.

Firstly, the ultimate setting is in the context of an interactive search—therefore rapid identification of promising solutions is important. The many conflicting desiderata and subjective choices make it unlikely that users will want to spend time interacting with a system that is "fine-tuning", preferring to do this by hand. It also means that population-based approaches are more likely to be suited to the multi-objective nature. For these interlinked reasons, although we consider a simple Local Search algorithm for the sake of completeness, we do not consider more complex approaches, such as Tabu Search and Simulated Annealing.

Secondly, viewing class-modelling as a partitioning task creates a combinatorial search space, ruling out some algorithms which are primarily designed for continuous domains or rely explicitly on computing fitness landscape gradients.

Thirdly, previous work by ourselves and others (Simons and Parmee 2010; Bowman et al. 2010) suggests that the software design search space is discrete, scattered and highly multi-modal. EAs have been used with some success for general "grouping" problems (Falkenauer 1998; Tucker et al. 2006), but there is still considerable debate over the best choice of representation to avoid massive redundancy in the phenotype/genotype mapping, and this remains a major unsolved problem (Lewis and Pullin 2011). Inherently the problem stems from the fact that, for example, if two elements i and j should be co-located within a class, then not only is the choice of label for that class irrelevant, but also adapting an EA to account for this representational constraint is non-trivial and at best creates a highly specialized algorithm. Given that the evolving population represents a probability distribution function of the assignment of elements to classes, one possibility might be to use an Estimation of Distribution Algorithm (EDA) (Lozano et al. 2006). For a graphical model that correctly captured the grouping above, then trivially the probabilistic model could evolve to look like $P(i = k|j = l) = \delta_{kl}$, where $\delta_{ij} = 1$ if $i = j$ and 0 elsewhere. However, currently EDAs, like other probabilistic model builders, use greedy search to construct models, so the search will at best be as effective as a greedy local search (GLC) algorithm in the space of partitions, hence we consider EAs, but not EDAs.

However, Ant Colony Algorithms (ACOs) (Dorigo and Stutzle 2004) have been used very successfully for problems with an inherent grouping component such as the Vehicle Routing Problem (VRP) (Toth and Vigo 2001) since the pheromone trail (broadly equivalent to the probabilistic graphical model in an EDA) can effectively contain a set of partial paths to be selected and traversed by ants without need for class labels, hence avoiding the whole issue of redundancy.

The search capability can be further enhanced if search specific generative heuristics can be exploited, and the population based nature of ACOs can be exploited for multi-objective problems (Lopez-Ibanez and Stutzle 2012; Cheng et al. 2012).

3.1 Representations

Two representations are used in this paper; both incorporate the object-oriented software design elements (i.e. methods, attributes) described in the previous section. Let c denote the maximum number of classes in the design solution, and d the number of design elements. In the first representation, which we shall call "naïve grouping" (NG), a candidate solution g is represented as a sequence of d integers from the set $\{1,\ldots,c\}$. Each integer in the sequence represents an individual software design element, so an assignment $g_i = j$ is interpreted as putting element i into class j. The search space is of size c^d, but there is considerable redundancy in the representation since the specific label applied to a class is irrelevant, so there are $d!$ equivalent representations of each design.

The second representation, which we shall call "Extended Permutation" (XP), is inspired by the Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) (Toth and Vigo 2001). Candidate solutions are represented as permutations of a set of $(d + c{-}1)$ elements, whereas above elements $\{1,\ldots,d\}$ represent the attributes and classes, and the final $c{-}1$ elements represent "end of class" markers. These are interpreted as akin to a "return to depot" in a VRP instance. Let i, j and k denote the positions in the permutation where the first three end-of-class markers occur, so that $0 < i < j < k < d + c{-}1$, and g_i, g_j, $g_k > d$, where g_i denotes the value of the *ith* position in the permutation representing solution g. The first class in the candidate design then contains the elements $\{g_1,\ldots,g_{i-1}\}$, the second the elements $\{g_{i+1},\ldots, g_{j-1}\}$, the third $\{g_{j+1},\ldots,g_{k-1}\}$, and so on.

This representation shares the redundancy of the NG, and adds to it by imposing a spurious order within each class that has no equivalent in the design space.

3.2 Fitness measures

To reflect the interactive nature of the meta-heuristic search, a combination of fitness measures is used. The measure of the structural integrity chosen is inspired by the "Coupling Between Objects" (CBO) measure (Harrison et al. 1998). Each candidate solution is decoded into a set of classes, and the CBO is calculated as the proportion of all uses of attributes by methods that occur across class boundaries. This is expressed as a maximization function $f_{CBO} = (1.0 - \mathrm{CBO}) \times 100$, so that $f_{CBO} = 100.0$ for a completely de-coupled design (all uses occur inside classes) and 0.0 for a completely coupled design.

However, it is also necessary to reflect the interaction of the designer within search. We have previously found that the elegance of the software design has proven to be a useful interactive measure (Simons and Parmee 2012), and proposed a number of quantitative elegance metrics relating to the evenness of distribution of attributes and methods among classes within the design. Building on this, two elegance metrics have been chosen as surrogates for human qualitative elegance evaluation, namely:

- *Numbers Among Classes* (*NAC*) the standard deviation of the numbers of attributes and methods among the classes of a design. This was truncated to the range [0,R] and a fitness to be maximised calculated as denoted $f_{NAC\cdot} = 100 * (\mathrm{R} - \mathrm{NAC})/\mathrm{R}$. The higher this value, the more symmetrical the appearance of attributes and methods among the classes in the design.
- *Attribute to Method Ratio* (*ATMR*) the standard deviation of the ratio of attributes to methods for each of the classes in a design. A fitness f_{ATMR} was calculated in the same way as above. The higher this value, the more even and symmetrical the appearance of this ratio across individual classes of the software design.

3.3 Evolutionary algorithm

The EA chosen for comparison uses deterministic binary tournaments for parent selection and a generational replacement model ensures the search is comparable to ACO. Random uniform mutation with either One-Point or Uniform crossover is applied to the NG representation. For the XP representation, we used Order-based crossover (Davis 1991) and "Edge Recombination" (Mathias and Whitley 1992).

The former preserves the relative order of elements (as per scheduling type problems) and the latter preserves adjacency information (as per TSP or VRP). These are standard operators from the literature. Full descriptions would take excessive space in this paper, but may be found in the freely available slides for genetic algorithms in the website supporting Eiben and Smith (2003).

We have previously shown that for many permutation-based problems the choice of mutation operator depends on both the problem instance and the state of the search (e.g. Krasnogor and Smith 2001; Serpell and Smith 2010 for adjacency-, and Smith et al. 2009 for order-based problems). Therefore, fixed mutation rates for the EA are interpreted as either the locus-wise probability of randomly resetting an allele value (NG) or as the probability of applying a single mutation event of type 'Swap', 'Insert' or 'Invert' chosen at random (XP). In both cases we also examined the utility of self-adaptation to provide robust optimization performance, and reduce the number of parameters required. Following the schemes in (Smith 2001; Stone and Smith 2002; Serpell and Smith 2010), a single extra gene is used to encode for one of a set of possible mutation rates. During mutation, first the encoded value is randomly reset with a fixed probability (the strategy_adaptation_rate), then a mutation event occurs in each locus with the encoded probability. This algorithm is described in pseudo-code in Fig. 2.

Depending on the representation chosen Recombine() calls one of the crossover operators listed above. For fixed mutation probabilities the parameter strategy_adaptation_rate is set to zero. For the XP representation the function Mutation() calls one of Swap, Insert or Insert mutation, chosen at random; for NG it calls random uniform mutation.

3.4 Ant colony optimization

Ant colony optimizations have been used successfully for permutation-type problems (Toth and Vigo 2001; Dorigo et al. 2006) where the pheromone trails map naturally onto path-based problems such as the TSP and VRP. Therefore, it was natural to use the XP representation described above. The ACO used in this paper is inspired by Dorigo and Stutzle (2004) and is described in pseudo-code in Fig. 3.

In the Initialize_Pheromone_Trails component, all trail pheromone values are initially set to 1.0. In subsequent iterations of the ACOMetaheuristic, pheromone values are not constrained, other than to ensure they remain ≥0.0. In the Construct_Ant_Solutions component, each ant creates a solution path by visiting elements (attributes, methods or "end of class") to traverse arcs (i, j) of the graph in turn, choosing each element probabilistically according to the value of pheromone trails τ_{ij} (laid down by previous ants) whose 'attractiveness' is controlled by raising it to the power of an attractiveness parameter, α. In the Deamon_Actions component, each solution path created is evaluated for fitness i.e. f_{CBO}, f_{NAC} and f_{ATMR}. Lastly, in the Update_Pheromones component, pheromone trail values are firstly evaporated

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}$$

where ρ_{ε} (0, 1] is a parameter. After pheromone evaporation has been applied to all arcs, an amount of pheromone is added to the arcs in proportion to previously determined fitness values. The amount of pheromone is controlled by an update parameter μ, to which the fitness value is raised prior to update.

3.5 Greedy local search

In keeping with the focus on re-using of-the-shelf components, GLS is implemented as (1 + 1) version of the EA code with mutation replaced by systematic (rather than randomized) search of the NG values or the 2-opt neighborhood for the XP representation.

Fig. 2 EA meta-heuristic in pseudo-code

```
Procedure EaMetaheuristic
      set parameters
      Initialize_Population
      while( termination_condition_not_met )
            set i = 0
            while( i < population_size )
                  parent1 = Binary_Tournament( population )
                  parent2 = Binary_Tournament( population )
                  offspring[i] = Recombine( parent1, parent2 )
                  if( rand() < stategy_adaptation_rate )
                        selection_new_mutation_rate(offspring[i])
                  Mutate( offspring[i] )
                  Evaluate( Offspring[i] )
            end while
            set population = offspring
      end while
end procedure
```

```
Procedure ACOMetaheuristic
      set parameters
      Initialize_Pheromone_Trails
      while( termination_condition_not_met )
            Construct_Ant_Solutions
            Deamon_Actions
            Update_Pheromones
      end while
end procedure
```

Fig. 3 ACO meta-heuristic in pseudo-code

4 Factors affecting choice of search engine

We identify a number of factors that we believe should be considered when choosing the meta-heuristic as part of creating an interactive tool. Some of these factors lead themselves to be easily quantified, other less so. Without wishing to second guess the uses to which interactive optimization can be turned, we do not attempt to rank these according to importance. Rather, we present them using examples of software design problems to illustrate the issues involved.

4.1 Scalability

When we consider scalability, we mean not just how well do the algorithms scale to solve larger problems, but how well can they assist the human in the process of designing large software solutions. Typically this process will involve partitioning the problem in some way to reduce its dimensionality. Key factors therefore include how well the algorithms support the user in first identifying good partial solutions, and then "freezing" those partial solutions. Clearly this depends on the complexity of the mapping from candidate solution (as presented to the user for evaluation) back to representation (as used by the search engine). For a software design, it is fairly simple to imagine an interactive box whereby a user can select a class to "freeze". For an EA, or Local Search based method, this requires some method for permanently recording the information that certain genes should not be affected by mutation/perturbation and should be co-transmitted under recombination. However, some meta-heuristics explicitly perform this partitioning process and so intrinsically record this information. Thus this "freezing" process can be instantiated by simply "fixing" some elements of the graphical model in an EDA or setting the relevant pheromone levels to an arbitrary high value in an ACO.

The other aspect of scalability for interactive meta-heuristic search is of course how well a given algorithm scales without the freezing process. Given the limits on human attention, this relates to the ability to discover high quality solutions in relatively few evaluations as the dimensionality of the problem increases. It is probably pointless to attempt to draw any firm conclusions about the relative scalability of different methods, as this is of course entirely problem and parameter dependent, although recent theoretical results (Birattari et al. 2007) suggest how to avoid poor scalability in ACO which was previously thought to be a problem. However, the need to make rapid advances in fitness rather than evaluating randomly created solutions points to either the use of local search, or to small populations in EAs or ACOs. EAs are known to work well with small populations (e.g. CMA-ES for continuous optimization), this is less well examined for ACOs. One major factor that should be considered is the number of human interactions required, and the availability (or otherwise) of surrogate fitness function that can reduce this.

4.2 Robustness

Meta-heuristic robustness relates to a number of factors e.g.

- *Appropriateness of representation* how sensitive and appropriate is the representation? For example, might a permutation representation cause problems of degeneracy?
- *Support for search* how well do the algorithms support for single- and multi-objective search?
- *Parameter choice* is algorithm performance effective across a range of parameters?
- *Parameter tuning/self-adaptation* can parameters be automatically tuned and/or controlled?

4.3 'Off-the-shelf' availability

A number of frameworks and toolkits for implementing EAs are readily and freely available, in all of the major programming languages and environments. Most of these can be adapted to run with a parent population of size one in order to implement local search. Well known toolkits such as Evolving Objects (Keijzer et al. 2002) and ECJ (Luke et al. 2012) implement a range of different data types and provide sufficient different mutation and recombination operators to give the user considerable flexibility in their choice of problem representation. Similarly, a number of implementations of the ACO meta-heuristics for optimization (rather than data mining) are available in C and C++ programming languages via the ACO website (2012). Although a range of different algorithm variants are supported and versions for different problems are available, all assume the use of a permutation representation for solutions. This restriction of the available ACO implementations, and the fact that most papers in this field deal with a path-type representation, would appear to rule out the straightforward use of ACOs for some problems. After consideration, all of the meta-heuristic approaches considered in this paper were implemented by the authors taking specifications from the literature. Assuming a reasonable knowledge of software engineering, the available implementations could be adapted to other problems (for example the heuristic rules used to initialize and augment the pheromone trails) but the level of documentation and support is not as comprehensive as might be expected from the relative maturity and popularity of the field—which should not be read to reflect the scientific merit.

4.4 Constraint handling

A crucial factor in the choice of meta-heuristic is the ease (or difficulty) with which the algorithm can handle any domain-specific constraints. For the software design problem, there is one crucial constraint as described earlier: each class must contain at least one method and one attribute. This constraint has a significant impact on the grouping of elements (attributes and methods) to classes, and the ease (or difficulty) with which the meta-heuristic copes with this is a key factor in its choice.

5 Methodology

5.1 Strategy

A key factor in the comparison of the meta-heuristic algorithms is the availability of plausible and representative test design problems for early lifecycle software design. Unfortunately, benchmark software design problems do not appear readily in either the research literature or industrial repositories. Therefore, three real-life software design problems have been selected for use. These have been chosen to provide an appropriate range of problem domain, and scale. While it is not possible to precisely assess how representative these might be of the software design field as a whole, both the second and third problems have been drawn from fully enterprise scale industrial software developments, and are decidedly non-trivial in size and complexity. Details of the three design problems are given in the following section, and full problem specifications are available on line (Simons 2012a, b, c).

To permit large scale comparisons we took a two-stage approach. Firstly, to establish the sensitivity of a meta-heuristic's performance to parameter values, the number of constraints, and how they are handled, we focussed on coupling between objects, optimizing f_{CBO} and comparing with manually produced designs. In this stage we used all combinations of the parameter values in Table 1.

Secondly, using the "best" parameter sets established for each method, we simulated multi-objective interactive search by introducing the surrogate elegance metrics into a weighted-sum approach and optimising: $f_{MO} = a. f_{CBO} + b. f_{NAC} + c. f_{ATMR}$. Empirical calibration revealed that with this linear model a weight of $a = 0.8$ was required for CBO, emphasising the importance of design structural integrity. To reflect the inherently noisy nature of human evaluation, we then chose b uniformly from the interval (0, $1-a$) and set $c = 1.0-a-b$.

All runs use a fixed number of classes—the same as in the manual design solution to provide comparability. To ensure repeatability of results, we made 50 runs for each test, i.e., each combination of algorithm, problem, encoding, and parameter values. Each run is allowed to continue until either one million solutions were evaluated, or a software design with fitness 100.0 was discovered. For each run we recorded the values of f_{CBO}, f_{NAC}, and f_{ATMR} for best solution found and the number of solutions evaluated before this best solution is first discovered. These are denoted MBF and AES, respectively.

Wherever results are analyzed by comparison of means, the "General Linear Model" of IBM SPSS Statistics tool version 19 is used with algorithm choice, population size and design problems as fixed factors, then applying ANOVA followed by post hoc testing using Tukey's "Honestly Significantly Different" test. In what follows, statements that effects are significant or not, should be read to mean that they are statistically significant with over 95 % confidence according to these tests.

5.2 Software design problems

Three software design problem domains are used as vehicles for investigation. The first is a generalized abstraction of a Cinema Booking System (CBS), which addresses, for example, making an advance booking for a showing of a film at a cinema, and payment for tickets on attending the cinema auditorium. A specification of the use cases of CBS design problem is available at Simons (2012a). The second problem is an extension to a student administration system to record and manage outcomes relating to the Graduate Development Program (GDP). This system was created by the in-house information systems department at the University of the West of England, UK in 2008. A specification of the use cases used in the development is available from Simons (2012b). The third software design problem domain is based on an industrial case study—select cruises (SC)—relating to a cruise company selling nautical adventure holidays. The resulting computerized system handles quotation requests, cruise reservations, payment

Table 1 Search parameters for meta-heuristic algorithms

	Parameter	Values trialled
Stochastic local search	Perturbation method	NG: allele-wise mutation
		XP: random one of insert/invert/swap mutation
Evolutionary algorithm	Selection method	Tournaments to select parents, size 2 and 5
		Generational replacement with elitism
	Crossover probability	0.0, 0.2, 0.4, 0.6, 0.8, 1.0
	Crossover operator	NG: uniform, one point
		XP: order-based, edge recombination
	Mutation probability	Self-adaptive: 1/*(0.001, 0.002, 0.01, 0.02, 0.01, 0.2, 1, 2, 5, 10)
		Fixed: 0.001, 0.01, 0.05, 0.1, 0.25, 0.5
	Strategy adaptation rate	0 (fixed)
		0.1 (self adaptive)
	Population size	25, 100
Ant colony optimisation	Trail attractiveness (α)	0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0
	Pheromone update (μ)	0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5
	Pheromone decay (ρ)	0.0, 0.01, 0.1, 0.25, 0.5, 1.0
	Ant colony size	25, 100

and confirmation via paper letter mailing. A specification of the use cases of Select Cruises design problem is available at Simons (2012c). Manual designs created by the appropriate software engineers for the three problems are available at Simons (2012d). Numbers of attributes, methods, classes, uses and f_{CBO}, f_{NAC} and f_{ATMR} (R = 6.0) for the manual designs are given in Table 2.

6 Single objective results

We begin this section with an assessment of the sensitivity of the two population based methods to their parameters. In general constraints can be handled directly (i.e. via repair mechanisms, specialized operators, or decoders) or indirectly via penalty functions (Eiben and Smith 2003). The former can be more efficient, but require problem specific alteration of the algorithms [ibid]. Therefore initially constraints were handled indirectly, by setting the fitness to zero for all solutions containing classes that had zero methods and/or zero attributes. This is followed by an analysis of a relatively minor change to each method so that new candidate solutions were regenerated (by following pheromone trials or via recombination and mutation) if they contained invalid classes—effectively a very crude direct method. We end by directly comparing the results from the "best" parameter sets of three meta-heuristics.

Table 2 Values of measures of manual software designs

	Atts	Meths	Classes	Uses	CBO	f_{CBO}	f_{NAC}	f_{ATMR}
CBS	15	16	5	39	0.154	84.6	86.31	96.69
GDP	43	12	5	121	0.297	70.3	56.80	56.38
SC	52	30	16	126	0.452	54.8	74.67	69.20

6.1 Greedy local search

Table 3 shows the results obtained with GLS under the two encodings. For all problems with XP, and for the SC problem with NG representation, some runs failed to find valid solutions, and the second column of fitnesses, and the AES column reflect only those runs funding valid solutions. Whether considering all, or only successful runs, the quality of solutions found by GLS algorithm is significantly higher on the NG landscape than on the XP one. There is also a far higher variance in the quality of local optima found on the XP landscape. GLS finds solutions with lower coupling than the manual design—but this perhaps merely emphasis the multi-objective nature of human design. It is worth commenting that the comparative values of AES on the two landscapes: values are typically an order of magnitude longer on NG than XP landscape, although of course this does not necessarily mean that good quality solutions are not discovered early on the way. We can understand this by noting that although both representations contain a certain amount of redundancy, there is considerably more for XP, so there are far more local optima in the XP landscape. Looking at the progress of sample runs, the number of solutions evaluated to reach given fitness values for XP and NG are 142/182 (fitness 50, CBS) and 480/839 (fitness 75 CBS), 491/491 (fitness 50, GDP) and 861/853 (fitness 75, GDP). These figures suggest that there is not a large difference in the initial rate of progress, but the GLS-

Table 3 Mean best coupling (MB) and average number of evaluations to best solution (AES) for greedy local search

	Manual f_{CBO}	Encoding	MB f_{CBO} *-all*	N valid	MB f_{CBO} *-valid*	AES
CBS	84.6	NG	87.25 (3.16)	50	87.25 (3.16)	62,669
		XP	62.35 (35.7)	38	82.04 (5.35)	5,272
GDP	70.3	NG	87.35 (4.05)	50	77.35 (4.05)	76,088
		XP	57.61 (36.46)	37	77.85 (13.80)	13,428
SC	54.8	NG	60.89 (27.1)	42	72.49 (4.07)	598,493
		XP	0.0 (0.0)	0	–	–

Results shown for all, and for just successful runs. Standard Deviation is shown in parentheses

NG subsequently explores for longer, finding higher quality solutions. On the SC problem the sample run of GLS-NG evaluated 372,006 solutions before finding a valid one and then a further 2,674 before finding one with fitness 50, i.e., comparable to the manual design. This suggests that as the number of classes increasing, the resulting constraints become a major factor in search effectiveness.

Overall, the high failure rates and variability of end results suggests that GLS-XP is unsuited to interactive search. The failure of some GLS-NG runs on SC shows that a specialized initialization operator is needed, and the subsequent long time to find a human-comparable design on SC, suggests that even if one is available, the GLS algorithm may be unsuitable for interactive search.

6.2 Evolutionary algorithms

Analysis of the results from EAs with both representations confirmed that the fixed mutation rate which gave the highest f_{CBO} values depended on both the problem (hence representation length) and on number of factors which affect the exploration–exploitation balance (population size, crossover operator and probability), and sub-optimal choices greatly deteriorated performance.

However, on a more positive note, results also clearly showed that for every problem-representation pairing, the use of self-adaptation leads to the discovery of solutions with significantly higher fitness than any of the fixed mutation rates, without any significant penalty in terms of the number of evaluations taken. Furthermore, when using self-adaptive mutation neither the choice of tournament size nor of crossover probability (within the range 0.2–0.8) made any significant difference to the f_{CBO} values. Since self-adaption not only yields superior results but also increases the EAs' robustness by reducing the number parameters required, for brevity we only report these results henceforth.

Figure 4 shows mean best coupling achieved with self-adapting mutation, 60 % probability of applying crossover for four crossovers and two population sizes. For the CBD and GDP problems there is no significant difference between population sizes, with either representation. Larger populations are beneficial for SC—not statistically significantly so for NG, but significantly so for XP. Analysis of run-logs suggests that this relates to the initialization problem as with GLS.

From the perspective of the robustness of the algorithms, there is a positive outcome in that the same settings do well across all three problems and on almost every run the EAs discover solutions with higher f_{CBO} than the equivalent human crafted solution. Only on the largest scale SC problem did EAs using the permutation representation fail to beat the human-crafted values.

Immediately apparent, and confirmed by statistical analysis, is that the NG (One point and Uniform recombination) representation leads to the discovery of better solutions than XP (Edge or Order recombination). For the SC problem, the difference is typically 50 %. Factoring out the effect of problem instance, there is not a significant difference between Uniform and One Point recombination for the NG representation. However, with the permutation-based XP, use of the Order crossover discovers higher quality solutions than Edge Recombination. Note that the latter preserves and transmits information about the frequency of co-occurrence of edges between nodes in good solutions—exactly the same information that the pheromone table encodes explicitly in the ACO algorithm.

Figure 5 shows progress of one typical run for each population size and problem with Order (XP) and Uniform (NG) crossover. As can be seen the smaller populations make more rapid progress in the initial stages and all algorithms continue to discover improved solutions long into their runs.

6.3 Ant colony optimization

An illustration of sensitivity of the mean best f_{CBO} values achieved for each design problem is shown in Fig. 6, for a value of ρ of 0.1 and a colony of 25 ants. To summarize the effects overall:

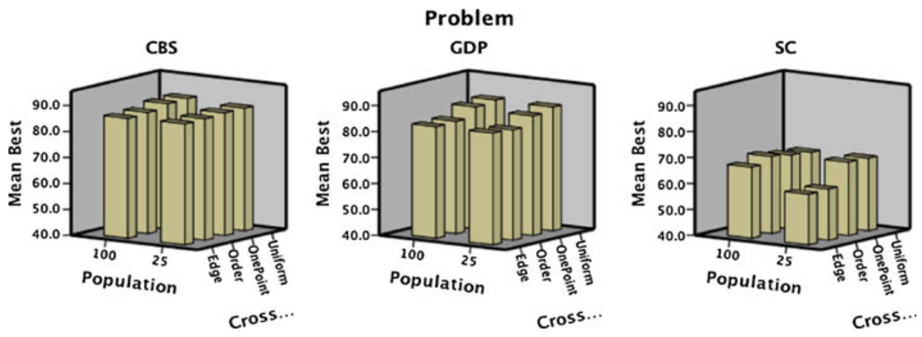

Fig. 4 Mean best coupling f_{CBO} achieved with EA using self-adaptive mutation, Px = 0.6. Edge and order crossover are for XP representation, uniform and one point for NG

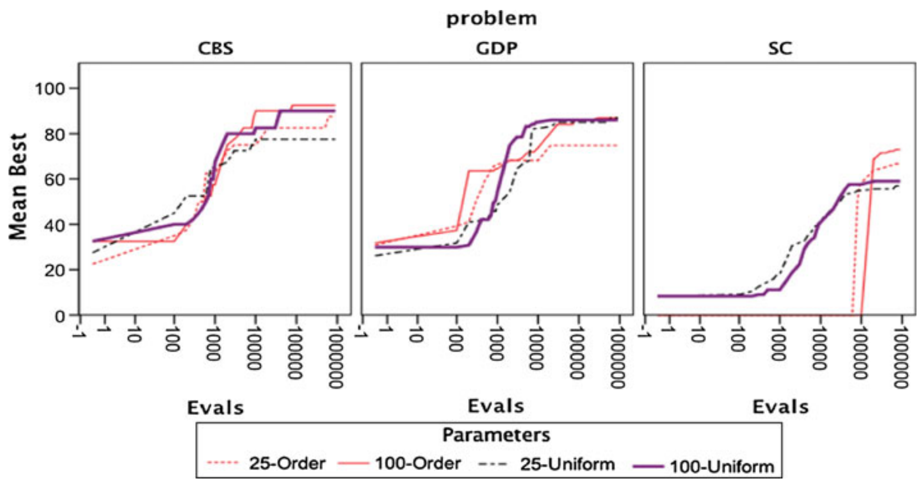

Fig. 5 Progress of typical EA runs. Note logarithmic scale of x-axis

- α: performance increases with α from 0 to 1.0–1.5 but tails off thereafter;
- μ: performance increases with μ from zero to 3.0;
- ρ: little effect for CBS and GDP, but for SC performance increases as ρ increases from 0 to 1.0;
- numbers of ants appear to have little discernible affect for CBS and GDP, but some effect on SC.

Figure 6 shows mean values, but the complexity of the response surface was confirmed by a multiple linear regression analysis, where the goodness of fit was greatly improved by extending the model to include quadratic and two-way interactions between α, μ and ρ for the three design problems.

In terms of the time taken to reach the best design solution, analysis shows that some degree of pheromone decay (i.e. $\rho > 0$) is necessary to achieve a plateau of fast performance (at higher values of α and μ). This plateau effect is visible with a ρ value of 0.01, but continues with increasing ρ to 1.0. This suggests that a degree of pheromone decay is crucial in exploiting the search space by making the algorithm able to 'forget' design solutions of poor fitness, especially if they are infeasible (i.e. do not contain at least one attribute and one method) (Fig. 7).

Therefore, considering mean best coupling and number of evaluations together—the balance between exploration and exploitation—a value of α of 1.0–1.5 appears to be effective when combined with some degree of pheromone decay ($\rho \geq 0.1$) and high values of pheromone update ($\mu = 3.0$) to balance the decay. Taking these values, Fig. 4 shows a typical ACO single run mean best f_{CBO}. It is observed that using 25 ants achieves mean best f_{CBO} quicker than using 100 ants, although for CBS and GDP, using 100 ants achieves a superior f_{CBO} after further evaluations.

Although ACO sensitivity to parameter values is undesirable from a robustness perspective, the "sweet spot' is the same for all three problems. With these settings, mean best f_{CBO} values of 86.17, 93.39 and 47.20 for CBS, GDP and SC respectively compare favorably those of the manual design for CBS and GDP, although noticeably less so for SC, which prompts further analysis.

6.4 Comparative analysis

Table 4 shows a comparison of the performance of the "best" version of the three search algorithms as identified above. The standard deviations are all in the range [0.5,5] except for GLS-XP (35–36) and GLS-NG on SC (27). Values in bold indicate the rankings per-problem according to the groups where the results are statistically significantly

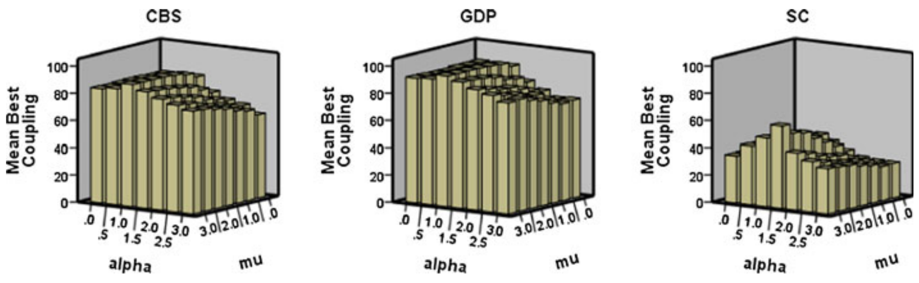

Fig. 6 Sensitivity of f_{CBO} values to parameters. 25 ants, $\rho = 0.1$

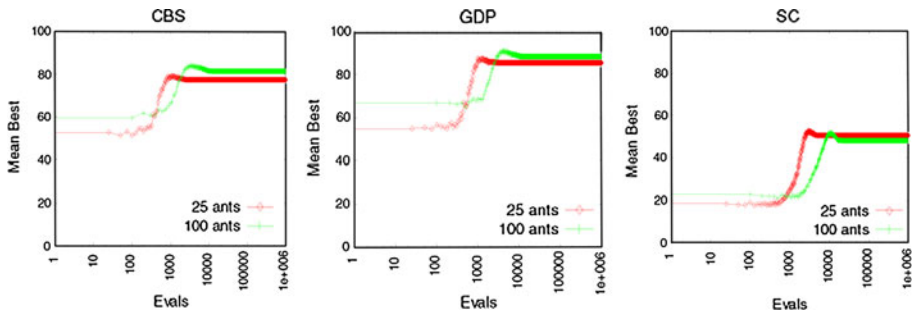

Fig. 7 Progress of typical ACO runs. Note logarithmic scale of x-axis

different. Given the budget of one million evaluations, it should be viewed as an indication of the ability of the algorithms to search landscapes induced by different representations for this problem. Since we have shown elsewhere that coupling is a correlated with human design judgment, if algorithms do not do well here, then there is little point subjecting humans to interacting with them. In most cases the metaheuristics create solutions with lower coupling than manual designs. Although GLS-NG does well on CBS and GDP, its low reliability on the SC appears to make it unsuitable for interactive applications.

These results reveal an interesting pattern. Although the NG landscape appears to be far more amenable to search by GLS or the EA, the results for the ACO are better (although not significantly so for 50 runs) despite the fact that the ACO is searching the far more multimodal and redundant working on the XP landscape. The exception to this is the SC problem. There are two possible reasons for this: the first is that the problem has more variables and that ACO does not scale. The second is that since it has more classes (16 as opposed to the 5 each for CBS and GSP) there are a far higher proportion of infeasible solutions, and the problem lies with the constraint handling.

6.5 Constraint handling

To investigate the worse behavior on the SC problem we ran ACO and EA-NG to produce designs with 5 classes rather than 16. In each case mean best f_{CBO} values of over 90 were observed—showing that the difficulty is one of the proportion of the search space that is infeasible, rather than its size.

Table 4 Comparison of mean best coupling (f_{CBO})

	Manual	GLS (XP)	EA (XP)	ACO	GLS (NG)	EA (NG)
CBS	84.6	62.35, **5**	82.10, **1=**	90.00, **1=**	87.25, **1=**	88.80, **1=**
GDP	70.3	57.61, **5**	77.46, **4**	96.20, **1=**	87.35, **1=**	88.07, **1=**
SC	54.8	0.0, **5**	42.68, **4**	49.76, **3**	60.89, **2**	67.74, **1**

Values in bold are statistically significant rankings

As described above, after an offspring solution (EA) or new solution path (ACO) is generated, a check is made to ensure that it contains at least one method and one attribute. In the indirect approach used so far, infeasible solutions are assigned a fitness 0.0. Although direct methods for constraint handling have been reported to be more effective (Eiben and Smith 2003), they would normally require highly specialized operators, sophisticated repair mechanisms which would mitigate against the use of these meta-heuristics as general purposes engines for interactive search. We therefore implemented the simplest form of "direct" approach in the EA and ACO—each newly created solution is checked, and regenerated until a feasible one is created. In the ACO the number of repeated is capped at 50.

Results with the ACO show that for the CBS and GDP problems the indirect approach leads to statistically significantly better results on the CBS, GDP and 5-class version of the SC problems, while for the 16-class SC the difference is not statistically significant. The differences between the numbers of evaluations to best solution are also not statistically significant. We hypothesize that direct

Table 5 Comparison of mean best coupling (f_{CBO}) with indirect and direct constraint handling

	EA-indirect-XP	EA-direct-XP	EA-indirect-NG	EA-direct-NG
CBS	82.10 (3.21), **4**	87.95 (1.31), **1=**	88.80 (2.49), **1=**	88.55 (2.58), **1=**
GDP	77.46 (2.75), **4**	83.10 (2.48), **3**	88.07 (3.69), **1=**	88.36 (3.44), **1=**
SC	42.68 (5.79), **4**	73.26 (2.09), **1=**	67.73 (5.41), **3**	71.92 (3.77), **1=**

Values in bold are statistically significant rankings

method performs less well because enforcing validity in early generations increases the probability of creating redundant versions of effectively the same design, which will confuse the process whereby the pheromone table adapts to model the structure of the underlying problem.

In contrast to this, as shown in Table 5, the beneficial results from implementing the direct mechanism in the EA are dramatic, especially as the scale of the problem, and hence the proportion of infeasible solutions increases, although notably, the less redundant NG representation still gives the best results.

7 Interactive search simulation

The results in Sect. 6 concerned the ability of the meta-heuristics to reliably locate good solutions for the software design problem given a large amount of evaluations. While this is useful to rule out some methods (e.g. GLS), the overall ranking of algorithms is only really relevant if a surrogate fitness model is available, since in practice humans can only evaluate a few dozens of solutions before fatigue and lack of engagement sets in. Thus we next compare the performance in a more time-limited scenario, leveraging our previous work to calculate fitness as a stochastic weighted sum of coupling and two elegance metrics to simulate the effect of user evaluation. Since we would not ask users to evaluate infeasible solutions, and initial experimentation showed that the effect of redundancy was much less with the smaller population sizes appropriate to visual display, we used direct constraint handling, i.e. infeasible solutions are re-created until valid.

Table 6 shows the results of running the three meta-heuristics ACO, EA-NG and EA-XP with population size 10, run for a maximum of 250 evaluations. The results shown are for the best individual found according to the f_{MO} metric averaged over 50 runs. The negative f_{ATMR} values for ACO generating solutions to SC arise from the way that the maximization function was calculated using a scale factor of R = 6.0: the algorithm repeatedly identified solutions with low coupling and high NAC elegance, but high variability between classes of the attribute-methods ratio amongst classes in the solutions found. The superior performance listed for on almost every metric of the ACO algorithm, is confirmed by ANOVA followed by Tamhane's post hoc test (since the variances are not equal).

While the coupling values obtained in a limited number of evaluations are inferior to the manual designs, the elegance values are comparable, and of course even 250 evaluations reflects less human effort than undertaking the manual design process. However, this does suggest that if it was intended to use f_{MO} as a surrogate fitness measure, it would be worth investigating a non-linear, or piece-wise linear function of f_{CBO} to place more emphasis on minimizing coupling.

8 Analysis

8.1 Scalability

In contrast to GLC, both population-based search algorithms (EAs and ACO) afford a good degree of scalability with respect to the numbers of elements (attributes and methods) present in the interactive software design problem. For the CBS and GDP design problems, both EAs and ACO discover design solutions of superior fitness compared to the hand-crafted, manual design, for both single-objective search and multi-objective interactive search simulation. Indeed, with high numbers of elements, ACO outperforms the EAs. However, the ACO struggled to achieve superior fitness with the scale of the SC design problem. Further analysis revealed that the crucial factor affecting scalability is the proportion of infeasible solutions in the search space, which reflects the constraint that each class must have at least one method and attribute. In the results above we have examined SC with 16 classes, to permit comparison with the manual design. However, the constraint could be relaxed by allowing completely empty classes. This would provide a means of designing solutions with variable numbers of classes, so avoiding the need to pre-specify this important aspect of the design structure.

With respect to 'freezing' of partial design solutions, both population-based search algorithms offer the opportunity to permanently record individual classes, or groups of classes. While the mechanisms for freezing (and unfreezing) parts of a design would require a major specialized adaptation of the EA, it is easily achieved in an ACO by simply setting the relevant pheromones to an arbitrary high value. Although not directly investigated here due to simulation difficulties, this may nevertheless be a fruitful tactic in any future interactive studies.

Table 6 Comparison of mean (std. dev.) results for best solutions found in simulation of interactive behaviour

Problem	Metric	EA-NG	EA-XP	ACO
CBS	f_{MO}	40.06 (5.79)	45.73 (5.61)	**60.35 (3.88)**
	f_{CBO}	49.80 (7.28)	**56.55 (7.57)**	48.26 (12.2)
	f_{NAC}	76.57 (6.32)	46.82 (10.49)	**99.30 (0.08)**
	f_{ATMR}	92.21 (4.13)	93.36 (3.89)	**97.34 (0.56)**
	AES	210 (45)	199 (54)	**140 (69)**
GDP	f_{MO}	31.73 (4.10)	44.20 (6.94)	**57.16 (3.21)**
	f_{CBO}	39.50 (5.13)	55.10 (8.68)	59.12 (11.15)
	f_{NAC}	65.27 (11.57)	4.46 (10.26)	**95.18 (0.77)**
	f_{ATMR}	70.82 (12.65)	69.91 (13.26)	**94.64 (0.55)**
	AES	230 (24)	222 (31)	**127 (74)**
SC	f_{MO}	13.36 (2.03)	15.04 (3.44)	**31.29 (3.41)**
	f_{CBO}	16.5 (2.55)	18.55 (4.25)	**39.25 (11.93)**
	f_{NAC}	79.85 (2.98)	73.328 (14.88)	**84.75 (2.01)**
	f_{ATMR}	**78.92 (4.62)**	76.57 (6.09)	−14.14 (91.32)
	AES	224 (38)	210 (51)	**110 (73)**

Bold type indicates statistically significant best per row, i.e., problem-metric combination

8.2 Robustness

In line with previous findings (Stone and Smith 2002; Serpell and Smith 2010), results of EAs show that self-adapting mutation enables a good degree of robustness, making the algorithm insensitive to crossover probabilities or selection pressure. Conversely, it is evident that ACO is sensitive to parameter values, although once tuned, the same set of parameter values ($\alpha = 1.0$–1.5, $\mu = 3.0$, $\rho = 0.1$) produces good results across all three software design problems. While the ACO graph representation appears more robust to varying numbers of elements in the software design solutions, the EA with integer-based naïve grouping representation appears more robust to varying numbers of classes.

8.3 ‘Off-the-shelf’ availability

As described in Sect. 4.3, key factors affecting the application of ‘off-the-shelf’ frameworks and toolkits for EAs and ACO appear to be (1) their ability to easily adapt to the specifics of the search problem at hand, and (2) available documentation and support. In the case of the software design problem, the specific constraint of the ‘at least one attribute and one method’ is not well catered for by the EA or ACO frameworks. In one sense this is not entirely surprising as software development frameworks are necessarily generic. In addition, available documentation, especially for ACO, is not sufficiently comprehensive. An additional although perhaps less important factor might be the choice of programming language provided by the framework. For example, ACO frameworks such as ACO (2012) focus on the C and C++ programming language which may or may not meet the portability and interoperability requirements of GUI-driven interactive meta-heuristic search engine.

8.4 Constraint handling

Handling the constraint of each class containing at least one attribute and at least one method is a significant constraint affecting the performance of all the meta-heuristic search algorithms investigated. A number of tactics to deal with constraint present themselves. Firstly, the constraint can simply be ignored in search but invalid design solutions attract zero fitness. Secondly, the search algorithm can be adapted and specialized to prevent the construction of invalid design solutions. Thirdly, the search algorithm can also be adapted to repeat construction of a design solution until a valid solution appears. Much of the investigation in this paper centers on the use of the first tactic, which appears to be effective. Indeed, perhaps surprisingly, results of Sect. 6.5 indicate that introducing constraint handling into the ACO algorithm produces inferior results from the point of view of an interactive design simulation. We hypothesize that this arises from the redundancy of the XP representation, and preliminary experimentation with decreasing population sizes for the MO search seemed to confirm our hypothesis.

9 Conclusions

This paper seeks to challenge the largely historical adoption of evolutionary computing as the basis of an engine for interactive search in early lifecycle software design. GLC, evolutionary algorithms and ACO have been compared and the performance of the population-based algorithms has been found superior to single-parent GLC. Indeed, experimental results show that population-based search produces software design solutions of fitness values superior to those of the hand-crafted design solutions.

Given a large computational budget—for example if a surrogate fitness model was available so only occasional user evaluation were required—the EA with the integer NG representation comes out as the clear favorite in terms of optimization performance. It is far more robust to parameter settings and the methods used for constraint handling.

However, if a wholly interactive search is required, thus small populations that can easily be visualized side-by-side, and a more limited computational budget, then a very different picture emerges. In this case the use of an ACO, with each ant forced to recreate its solution path until valid, emerges as finding higher quality solutions, and in around half the time of the EAs. Moreover, the simple modifications that would be required to allow user-friendly modifications such as the ability to "freeze" certain classes, or coalesce others also point to the use of interactive ACOs for larger problems.

References

(2012) ACO Meta-heuristic. http://www.aco-metaheuristic.org/aco-code/. Accessed 20 May 2012

Acampora G, Cadenas JM, Loia V, Ballester EM (2011) Achieving memetic adaptability by means of agent-based machine learning. IEEE Trans Indust Informat 7(4):557–569

Acampora G, Loia V, Salerno S, Vitiello A (2012) A hybrid evolutionary approach for solving the ontology alignment problem. Int J Intell Sys 27(3):189–216

Xanthakis S et al (1992) Application of genetic algorithms to software testing. In: Proceedings of the 5th Int'l Conf Softw Eng (ICSE 92), pp 625–636

Luke S et al (2012) ECJ 20: a java-based evolutionary computation research system. http://cs.gmu.edu/∼eclab/projects/ecj/. Accessed 20 May 2012

Al Dallal J, Briand LC (2010) An object-oriented high-level design-based class cohesion metric. Info Softw Tech 52(12):1346–1361

Avigad G, Moshaiov A, Brauner N (2005) Interactive concept-based search using MOEA: the hierarchical preference case. Intl J Comput Intell 2(3):182–191

Badillo AR, Ruiz JJ, Cotta C, Fernandez-Leiva AJ (2013) On user-centric memetic algorithms. Soft Comput 17(2):285–300

Birattari M, Pellegrini P, Dorigo M (2007) On the invariance of ant colony optimization. IEEE Trans Evol Comput 11(6):732–742

Booch G (1994) Object-oriented analysis and design, 2nd edn. Benjamin/Cummings Publishing, Redwood City

Booch G, Rumbaugh J, Jacobson I (1999) The unified modeling language user guide. Addison-Wesley, Boston

Boudjeloud L, Poulet F (2005) Visual interactive evolutionary algorithm for high dimensional data clustering and outlier detection. PAKDD, Lecture Notes in Artificial Intelligence, pp 428–431

Bowman M, Briand LC, Labiche Y (2010) Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. IEEE Trans Softw Eng 36(6):817–837

Briand LC, Daly JW, Wust JK (1999) A unified framework for coupling measurement in object-oriented systems. IEEE Trans Softw Eng 25(1):91–121

Brintrup A, Ramsden J, Takagi H, Tiwari A (2008) Ergonomic chair design by fusing qualitative and quantitative criteria using interactive genetic algorithms. IEEE Trans Evol Comput 12(3): 343–354

Caldwell C, Johnston VS (1991) Tracking a criminal suspect through "Face-Space" with a genetic algorithm. In: Proceedings of the 4th International Conference on Genetic Algorithms, pp 416–421

Caleb-Solly P, Smith J (2007) Adaptive surface inspection via interactive evolution. Image Vision Comput 25(7):1058–1072

Cheng J, Zhang G, Li Z, Li Y (2012) Multi-objective ant colony optimization based on decomposition for bi-objective travelling salesman problems. Soft Comput 16(4):597–614

Davis L (ed) (1991) Handbook of genetic algorithms. Van Nostrand Reinhold, New York

Dawkins R (1990) The blind watchmaker. Penguin Books, Harmondsworth

Dorigo M, Stutzle T (2004) Ant colony optimisation. MIT Press, Cambridge

Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intel Mag 1(4):28–39

Eiben AE, Smith JE (2003) Introduction to evolutionary algorithms. Springer. Supporting website with slides for operator descriptions at http://www.bit.uwe.ac.uk/%7Ejsmith/ecbook/slides/. Accessed Feb 2013

Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, New York

Geiger MJ (2008) Proposition of the interactive pareto iterated local search procedure—elements and initial experiments. Submitted on 4 September 2008. http://arXiv.org

Harman M (2007) The current state and future of search based software engineering. In: Proceedings of Future of Software Engineering. FOSE '07, pp 342–357

Harman M (2011) Software engineering meets evolutionary computation. Computer 44(10):31–39

Harman M, Jones BJ (2001) Search-based software engineering. Info Softw Tech 43(14):833–839

Harman M, Tratt L (2007) Pareto optimal search-based refactoring at the design level. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07), pp 1106–1113

Harrison R, Councell S, Nithi R (1998) An investigation into the applicability and validity of object-oriented design metrics. Emp Softw Eng 3(3):255–273

Jones BF, Sthamer H–H, Eyres DE (1996) Automatic structural testing using genetic algorithms. Softw Eng J 11(5):299–306

Keijzer M, Merelo JJ, Romero G, Schoenauer GM (2002) Evolving objects: a general purpose evolutionary computation library. Artif Evol 23(10):829—888. http://eodev.sourceforge.net/. Accessed May 2012

Kopfer H, Schonberger J (2002) Interactive solving of vehicle routing and scheduling problems: basic concepts and qualification of tabu search approaches. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02), pp 1425–1434

Krasnogor N, Smith JE (2001) Emergence of profitable search strategies based on a simple inheritance mechanism. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '01), pp 432–439

Kubota N, Nojima Y, Kojima F, Fukuda T (2006) Multiple fuzzy state-value functions for human evaluation through interactive trajectory planning of a partner robot. Soft Comput 10(10): 891–901

Lee J-Y, Cho S-B (1999) Interactive genetic algorithm with wavelet coefficients for emotional image retrieval. In: Proceedings of the 5th International Conference on Soft Computing and Information/Intelligent Systems, vol 2, pp 829–832

Legrand P, Bourgeois-Republique C, Pean V, Harboun-Cohen E, Levy-Vehel J, Frachet B, Lutton E, Collet P (2007) Interactive evolution for cochlear implants fitting. Gen Prog Evol Mach 8(4):301–318

Lewis R, Pullin E (2011) Revisiting the restricted growth function genetic algorithm for grouping problems. Evol Comput 19(4): 693–704

Lopez-Ibanez M, Stutzle T (2012) An experimental analysis of design choices for multi-objectives ant colony optimisation algorithms. Swarm Intel 6(3):207–232

Lozano P, Larranga P, Inz I, Bengoetxea E (eds) (2006) Towards a new evolutionary computation: advances in estimation of distribution algorithms. Springer, Berlin

Madar J, Abonyi J, Szeifert F (2005) Interactive particle swarm optimisation. In: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications (ISDA'05), pp 314–319

Mathias K, Whitley D (1992) Genetic operators, the fitness landscape and the traveling salesman problem. In: Proceedings of Parallel Problem Solving from Nature (PPSN'92), pp 219–228

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

Miller G (1956) The magical number seven, plus or minus two: some limits on our capacity for processing information. Psych Rev 63(2):81–97

O'Keeffe M, Cinneide MO (2008) Search-based refactoring for software maintenance. J Sys Softw 81(4):502–516

Object Management Group (2012) Unified modelling language resource page. http://www.uml.org/. Accessed 12 April 2012

Ohsaki M, Takagi H, Ohya K (1998) An input method using discrete fitness values for interactive GA. J Intel Fuzzy Syst 6(1): 131–145

Pauplin O, Caleb-Solly P, Smith J (2010) User-centric image segmentation using an interactive parameter adaptation tool. Pattern Recogn 43(2):519–529

Ren J, Harman M, Di Penta M (2011) Cooperative co-evolutionary optimisation of software project assignments and job scheduling. In: Proceedings of the 3rd International Symposium of Search Based Software Engineering (SSBSE 2011), Lecture Notes in Computer Science, vol 6956, pp 127–141

Serpell M, Smith JE (2010) Self-adaption of mutation operator and probability for permutation representations in genetic algorithms. Evol Comput 18(3):1–24

Simons CL (2011) Interactive evolutionary computing in early lifecycle software engineering design. PhD Thesis, University of the West of England, Bristol

Simons CL (2012a) Use case specifications for cinema booking system. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/CinemaBookingSystem.htm. Accessed 20 May 2012

Simons CL (2012b) Use case specifications for graduate development program. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/GraduateDevelopmentProgram.htm. Accessed May 2012

Simons CL (2012c) Use case specifications for select cruises. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/SelectCruises.htm. Accessed May 2012

Simons CL (2012d) Manual software designs for problem domains. http://www.cems.uwe.ac.uk/~clsimons/CaseStudies/ManualDesigns.pdf. Accessed May 2012

Simons CL, Parmee IC (2010) Dynamic parameter control of interactive local search in UML software design. In: Proceedings of the 2010 International Conference on Systems, Man and Cybernetics (SMC'10), pp 3399–3904

Simons CL, Parmee IC (2012) Elegant object-oriented software design via interactive evolutionary computation. IEEE Trans Systems Man Cybern Part C 42(6):1797–1805

Simons CL, Parmee IC, Gwynllyw R (2010) Interactive, evolutionary search in upstream object-oriented class design. IEEE Trans Softw Eng 36(6):798–816

Sims K (1991a) Interactive evolution of dynamical systems. First European Conference on Artificial Life, MIT Press

Sims K (1991b) Artificial evolution for computer graphics. Comp Graph (Siggraph '91 Proceedings) 25(4): 319–328

Smith JE (2001) Modelling GAs with self-adaptive mutation rates. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01), pp 599–606

Smith JE, Fogarty TC (1996) Evolving software test data: GAs learn self- expression. In: Fogarty TC (ed) Evolutionary computing. Springer, Berlin, pp 137–146

Smith JE, Bartley M, Fogarty TC (1997) Microprocessor design verification by two-phase evolution of variable length tests. In: Proceedings of the 1997 IEEE Conference on Evolutionary Computation, pp 453–458

Smith JE, Clark A, Staggemeir A (2009) A genetic approach to statistical disclosure control. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computing (GECCO'09), pp 1625–1632

Stone C, Smith JE (2002) Strategy parameter variety in self-adaptation of mutation rates. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02), pp 586–593

Takagi H (2001) Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. Proc IEEE 89(9):1275–1298

Takagi H, Ohsaki M (2007) Interactive evolutionary computation-based hearing-aid fitting. IEEE Trans Evol Comput 11(3): 414–427

Toth P, Vigo D (2001) The vehicle routing problem. SIAM, Philadelphia

Tucker A, Crampton J, Swift S (2006) RGFGA: an efficient representation and crossover for grouping genetic algorithms. Evol Comput 13(4):477–499

Uğur A, Aydin D (2009) An interactive simulation and analysis software for solving TSP using ant colony optimization algorithms. Adv Eng Softw 40(5):341–349

Weimer W, Forrest S, Le Goues C, Nguyen T (2010) Automatic program repair with evolutionary computing. Comm ACM 53(5):109–116

Wirfs-Brock R, McMean A (2003) Object design: roles, responsibilities, and collaborations. Addison-Wesley, Boston

Zhang Y (2012) Repository of publications on search-based software engineering. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/. Accessed April 2012