A Comparison of Evolutionary Algorithms and Ant Colony Optimization for Interactive Software Design

Christopher Simons and James Smith

Department of Computer Science and Creative Technologies
University of the West of England, Bristol BS16 1QY, United Kingdom

{chris.simons, jim.smith}@uwe.ac.uk

Abstract. Evolutionary algorithms have a well-documented history of success in Search-Based Software Engineering, but arguably the preponderance of evolutionary algorithms stems from 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 object-oriented software design. We consider a range of factors that should affect the design choice including representation, scalability, and of course, performance, i.e. that ability to generate good solutions within the limited number of evaluations. We then evaluate Evolutionary Algorithms and Ant Colony Optimization with a variety of representations for candidate solutions. Results show that after suitable parameter tuning, Ant Colony Optimization is highly effective 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 an integer-based representation appears more scalable.

Keywords: Ant Colony Optimization, Evolutionary Algorithms, Software Design

1 Introduction

Search-Based Software Engineering (SBSE) is now a well-established discipline wherein search has been applied across the range of the software development lifecycle [1]. Increasing research focus has been directed recently to the upstream stages of the software design, where meta-heuristic search of design spaces such as the object-oriented modeling of design classes has used metrics relating to coupling and cohesion as fitness functions [2], [3]. However, in the early stages of the software development lifecycle, the software designer has many competing factors to balance. In this situation, the role of search is therefore to enable the exploration of the search
space to discover useful and relevant software designs, and so provide insight into the
design task at hand. Here the precise balance of factors affecting the subjective judg-
ments of the human software designer 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 of Takagi [4]). Our earlier work demonstrates that we can
indeed successfully use meta-heuristics to provide computational support for an inter-
active software design process, evolving object-oriented class models that meet de-
signers’ criteria – both subjective [5] and aesthetic [6].

As with most papers in the field, such interactive design search uses an Evolution-
ary Algorithm (EA) [7] because of their long history of successful applications. How-
ever, as the name of the field of Search-Based Software Engineering suggests, poten-
tially any search algorithm could be used, although in practice research effort has also
tended to concentrate on meta-heuristics, in particular Evolutionary Algorithms. 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 re-
searchers have adopted them for SBSE problems; none of these are scientific motiva-
tions for adoption.” [8].

The contribution of this paper is to compare evolutionary algorithms with ant col-
ony optimization in order to make a more informed choice for an underlying search
engine for interactive search of the object-oriented software design search space. Sec-
tion 2 outlines object-oriented software design, while section 3 describes the search
algorithms investigated. The experimental methodology is described in section 4 and
the results of investigations are revealed in section 5. Finally in section 6, we con-
clude by contrasting the performance characteristics of evolutionary algorithms and
ant colony optimization in object-oriented software design.

2 Object-Oriented Software Design

Software design involves the identification of concepts and information relevant to
the design problem domain under investigation. Using the object-oriented paradigm,
such concepts and information are expressed using the ‘class’ construct, where indi-
vidual instances of classes are known as objects. These classes and objects have cru-
cial relevance to subsequent downstream software implementation and testing. The
Unified Modeling Language (UML) [9] is the standard modeling language of the
object-oriented paradigm, and is widely used by software designers to visualize and
specify classes as well as other aspects of software designs. Using the UML, classes
are 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
or indeed with human users, other programs etc.) Thus early lifecycle software design
involves finding an appropriate grouping of attributes and methods into classes.
3 Search Algorithms

Two candidate representations have been chosen to investigate search of the software design search space. In the first representation, which we shall call “naïve grouping” (NG), the genotype is a sequence of $d$ integers from the set \{1,...,$c$\}, (where $c$ is the maximum number of classes allowed), with an allele value of grouping $g_i = j$ being interpreted as putting element $i$ into class $j$. The search space is of size $c^d$, but there is considerable redundancy in the genotype-phenotype mapping since as far as the class model is concerned; the label applied to a class is irrelevant. The second representation is that of a graph inspired by the Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) [10] which we shall call Extended Permutation (XP). In this graph, candidate solutions are represented as permutations of a set of $(e+c)$ elements, where $e$ are the attribute and method elements and the extra $c$ elements are interpreted as “end of class” markers akin to a “return to depot” in a VRP instance.

It is generally understood in software engineering that designers strive for high cohesion in classes (to reflect a clear purpose) and low coupling between objects (to ensure the design is robust yet flexible to change). Therefore, the measure of the structural integrity chosen is inspired by the “Coupling Between Objects” (CBO) measure [11]. Regardless of the representation chosen, 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 $f_{CBO} = (1.0 - \text{CBO}) \times 100$, so that $f_{CBO} = 100.0$ for a completely decoupled design (all uses occur inside classes) and 0.0 for a completely coupled design.

The evolutionary algorithm chosen for comparison uses deterministic binary tournaments for parent selection and a generational replacement model ensuring the search is comparable to ACO. To the NG representation is applied random uniform mutation with either One-Point or Uniform crossover. For the XP representation, we used Order-based crossover [12] and “Edge Recombination” [13]. The former preserves the relative order of elements (as per scheduling type problems) and the latter preserves adjacency information (as per TSP or VRP). Following the scheme in [14] 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 probability 0.1, and then a mutation event occurs in each locus with the encoded probability.

The ant colony optimization has been implemented as described in [15] and uses the XP representation described above. Each ant creates a solution by visiting elements (attributes, methods or “end of class”) in turn, choosing each element probabilistically according to a combination of the attractiveness ($\alpha$) of pheromone trails (laid down by previous ants) and heuristics. After the whole population (colony) has created tours, all pheromone trails are subject to evaporation at a constant rate ($\sigma$). Finally, for each link traversed in each of the trails, a small amount ($\mu$) of additional pheromone is laid down proportional to the fitness of the trail in which it occurred.
4 Methodology

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. 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. The first test design problem relates to a cinema booking system (CBS), and comprises 16 attributes and 15 methods. The second test design problem relates to the Graduate Development Program (GDP) recording system deployed at the University of the West of England, UK, and comprises 43 attributes and 12 methods. Lastly, the third design problem relates to a nautical holiday booking system, Select Cruises (SC), which comprises 52 attributes and 30 methods. Specifications of the three test design problems can be found at [16]. Manual, hand-crafted designs are available for the three test design problems [17], and \( f_{CBO} \) values are 84.6%, 70.3% and 54.8% for CBS, GDP and SC respectively.

Parameter values are drawn from the literature (e.g. [7], [15]). All search runs use a fixed number of classes – the same as in the manual design solution to provide comparability. Manual designs comprise 5, 5 and 16 classes for CBS, GDP and SC respectively. 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 fitness (the values of \( f_{CBO} \) for best solution found) and speed (the number of solutions evaluated before this best solution is first discovered).

5 Results

5.1 Evolutionary Algorithms

Results showed that for every problem-representation pairing, the use of self-adapting mutation 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.

Results reveal and statistical analysis confirms 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.
5.2 Ant Colony Optimization

Results of parameter tuning are as follows:

- $\alpha$: performance increases as $\alpha$ increases from 0 to 1.0 – 1.5 but tails off thereafter;
- $\mu$: performance increases as $\mu$ increases from zero to 3.0;
- $\rho$: little effect for CBS and GDP, but for SC performance increases as $\rho$ increases from 0 to 1.0.

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 fast performance (at higher values of $\alpha$ and $\mu$). 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.

5.3 Comparative Analysis

After parameters have been tuned, Table 1 shows a comparison of the $f_{CBO}$ fitness of the three search algorithms while table 2 shows the number of generations / iterations required to achieve the $f_{CBO}$ values stated in table 1. Both tables relate to a population / colony size of 100. In most cases the meta-heuristics create solutions with lower coupling than manual designs. Interestingly, the NG landscape appears to be more amenable to EA search than XP, yet ACO produces better fitness values for CBS and GDP but not SC. This suggests that the EA with the NG landscape is more robust to increased scale in terms of the number of classes in the design.

**Table 1. Comparison of Mean Best Coupling ($f_{CBO}$).**

<table>
<thead>
<tr>
<th>Design Problem</th>
<th>Manual Design</th>
<th>EA (XP)</th>
<th>EA (NG)</th>
<th>ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBS</td>
<td>84.6%</td>
<td>82.10%</td>
<td>88.80%</td>
<td>90.00%</td>
</tr>
<tr>
<td>GDP</td>
<td>70.3%</td>
<td>77.46%</td>
<td>88.07%</td>
<td>96.26%</td>
</tr>
<tr>
<td>SC</td>
<td>54.8%</td>
<td>42.68%</td>
<td>67.74%</td>
<td>49.76%</td>
</tr>
</tbody>
</table>

**Table 2. Comparison of the Number of Generations / Iterations required.**

<table>
<thead>
<tr>
<th>Design Problem</th>
<th>EA (XP)</th>
<th>EA (NG)</th>
<th>ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBS</td>
<td>400</td>
<td>350</td>
<td>27</td>
</tr>
<tr>
<td>GDP</td>
<td>350</td>
<td>200</td>
<td>34</td>
</tr>
<tr>
<td>SC</td>
<td>1000</td>
<td>800</td>
<td>90</td>
</tr>
</tbody>
</table>
6 Conclusions

Comparison of evolutionary algorithms and ant colony optimization reveals that with respect to solution fitness, both approaches produce software design solutions of fitness values superior to those of the hand-crafted design solutions, except ACO for the SC problem. The EA with the integer NG representation emerges as the clear favorite in terms of performance for high numbers of classes, and with self-adaptive mutation, is far more robust to parameter settings. However, if a wholly interactive search is required for designs with smaller numbers of classes but higher numbers of attributes and methods and a limited computational budget, then a very different picture emerges. In this case, ACO discovers higher quality solutions, and in less time than the EAs.

References