A comparison of Genetic Algorithms and Evolutionary Programming in Conceptual Software Design

C. L. Simons, I. C. Parmee
Advanced Computation in Design and Decision Making,
Faculty of Computing, Engineering and Mathematical Sciences,
University of the West of England,
Bristol, BS16 1QY, UK

+44.117,3283135, +44.117.3283137

{chris.simons,ian.parmee}@uwe.ac.uk

Abstract

In an attempt to support the software engineer during conceptual software design, the authors have proposed that the benefits of engineering design search-based approaches may be applied to conceptual software design and have previously described the characteristics of the conceptual software design search space. This paper investigates the exploration of this space via a comparison of Genetic Algorithm (GA) and Evolutionary Programming (EP) search processes. Using the software engineering concept of cohesion as an objective fitness criterion, the average population cohesion fitness of both GA and EP has been found to be similar to the cohesion value of a human-performed design for a case study. However, differences in the characteristics of GA and EP were evident. GA has been found to converge on local optimal designs faster than EP while EP has been found to produce a greater variation of designs. We conclude that it is unlikely that the large number and huge diversity of optimal conceptual design variants could have been produced manually be the software designer. Based on this, we suggest that this mass of locally optimal design variants offers significant potential to support the designer when integrated with user-centric, computationally intelligent tools.

1. Introduction

Empirical evidence suggests that the process of conceptual software design is difficult for human designers to perform [1-3]. The negative impact and cost of these difficulties to software engineering is difficult to judge, but poor conceptual designs may have significant deleterious down-stream consequences for software development. A previous survey by the authors [4] suggested a mixed picture of success for conceptual software design.

Search-based approaches to design have received significant attention within the evolutionary computing (EC) domain, including applications in engineering design [5]. In particular, Parmee has reported the successful application of evolutionary computing to improve problem definition through an interactive approach with the designer [6], and also human-centric intelligent systems for exploration and knowledge discovery [7]. In a cross disciplinary transfer of technology, the authors have proposed that the benefits of an engineering design search-based approach may be applied to software engineering conceptual design. To support this, preliminary work by the authors has described the characteristics of a search space for conceptual software design [8]. To advance understanding of search and exploration of the conceptual software design space, this paper evaluates the potential of two evolutionary algorithms when compared with human–performed design.

2. Related Work

Applying search space exploration and optimization to software engineering was proposed by Harman and Jones in 2001 [9], who suggested that these techniques might be applied to various stages of the software development life cycle. Search-based approaches were outlined alongside other applications of computational intelligence to software engineering by Lee [10]. A recent example of work relating to search-based approaches to software engineering can be found at [11] where software clustering and subsystem decomposition were investigated.

The application of search-based approaches to software engineering is currently being researched in all phases of the software development lifecycle. For example, the clustering and grouping algorithms reported above are applied downstream after the human designer has designed and built the software. However, this differs significantly from the approach suggested in this paper where computational search and exploration supports the human designer during the process of conceptual design. This is consistent with other upstream approaches of search-based evolutionary algorithms within software engineering including software application design based on architecture, reliability and cost [12], and the application of clustering techniques to software component architecture design [13].

3. Representation

For effective search and exploration, it is necessary to represent both the design problem and the design solution search space. Both have been described previously by the authors in [8], and are summarised as follows. The design problem is represented as use cases [14], a technique widely used in software engineering. From the textual narrative of a use case, it is possible to identify actions that the software is to perform, and individual data items, which the software-to-be will

manipulate. If an action and a data item are co-located in a step of the narrative, the action is said to "use" the data. Thus the problem domain can be specified by three finite sets:

- a set of unique atomic data,
- a set of unique actions, and
- a set of uses.

The conceptual software design search space is then comprised of:

- A set of methods, directly derived from the set of actions of the design problem, and
- A set of attributes, directly derived from the set of data of the design problem.

The notion of a class is then introduced to the search space as a placeholder for the methods and attributes, such that each class must contain at least one method and at least one attribute.

The resulting representation of the search space is highly discrete. The class acts as the building-block of the search space, effectively grouping methods and attributes into classes as an object-based representation.

4. Fitness

The software engineering notion of cohesion has been used as the objective criterion to evaluate the fitness of conceptual software engineering designs. The Cohesiveness of Methods (COM) [15] metric has been used to determine cohesion of individual conceptual designs.

According to its proposers, Cohesiveness of Methods (COM) is: *"for each attribute, the sum of all the methods using an attribute divided by the total number of methods, all divided by the number of attributes in the class"*. Usefully, values of COM lie in the range 0 to 1. If all the attributes of a class are used by all the methods in that class, the value of COM is 1. On the other hand, if all the attributes of a class are used by none of the methods, the value of COM is 0. Where some of the attributes of a class are used by some of the methods of that class, the value of COM lies between 0 and 1. At conceptual design level, the COM fitness of a given design has been arrived at by averaging the COM fitness values of all classes within a design.

5. Evolutionary algorithms

The evolutionary algorithms investigated are in this paper are genetic algorithm and evolutionary programming. Although the genetic algorithm enables an evolutionary search process of the discrete software design space, preliminary findings suggested limitations in the crossover operator. Thus an evolutionary programming approach has also been developed to compare its balance of exploration versus exploitation with that of the genetic algorithm

5.1 Genetic Algorithm (GA)

The genetic algorithm used in this investigation is loosely derived from the simple binary string GA example presented in [16] and reiterated in [17]. Initialization of the population has been achieved by allocating a number of classes to each individual design at random, within a range derived from the number of use cases. Allocating the number of classes at random was necessary (and desirable from a search point of view) as the "correct" number of classes cannot be known *a priori*. All attributes and methods from the sets of attributes and methods are then allocated to classes within individuals at random.

Selection is performed by two techniques, namely tournament selection and roulette-wheel selection. Both techniques are derived from the simple GA example presented in [16] and [17]. Recombination is achieved by means of a trans-positional crossover (TPX) in which two individuals are chosen at random from the population, and their attributes and methods swapped between the two based on their class position in the individuals. For example, if an attribute was found to be in the first class of the first individual and the last class of the second, the attribute was relocated to the last class for the first individual, and the first class for the second. However, a constraint of the search space is that each class must contain at least one attribute and method. Thus positional swapping can only occur where swapping an attribute or method to another class would not leave the class lacking attributes or methods.

The approach taken to mutation has again been derived by analogy from the simple binary-string GA [16] where a single piece of genetic information is changed at random. Thus a single individual is mutated by relocating one attribute and one method from one class to another. GA mutation also complies with the 'at least one attribute and one method' constraint in that attributes / methods are not taken from classes with only one attribute / method.

5.2 Evolutionary Programming (EP)

The algorithm for EP used in this investigation is loosely derived from [18-20]. Initialization of the population is achieved by the same mechanism as used in the GA. For selection, each individual took part in COM fitness tournaments against 20 individuals selected at random from the population. A score was awarded to each individual based on the number of tournaments won, and each individual was ranked in the population according to their score. Highest ranking individuals progressed to the next generation.

Consistent with Fogel [18], recombination has not been performed in EP. Mutation has been performed by two mechanisms: class level mutation and element level mutation. At class level, all attributes and methods of a class in an individual were swapped as a group with another class selected at random. At element level, all elements (attributes and

methods) in an individual were swapped at random from one class to another, with the following exception. Element level EP mutation, like GA mutation, was limited by the 'at least one attribute and one method per class' constraint. Neither attributes nor methods were swapped where removing an attribute or method from the class would leave without an attribute or method.

6. Case Study

6.1 Cinema Booking System

The case study selected for investigation of EA search is a generalized cinema booking system, derived from a number of established internet-based cinema booking systems existing in the UK. This case study has been used by the first author as a software design problem for second year undergraduate software engineering students on many occasions. The design problem of the cinema booking system addresses, for example, making an advance booking for a showing of a film, and the collection of tickets on attending the cinema auditorium.

6.2 Human-performed conceptual design

Although varying conceptual designs for the cinema booking system have been put forward by many students and experienced software engineers over a number of years, the first author has noted that a number of conceptual abstractions tend to be repeatedly identified. Using methods and attributes derived from the problem space, an amalgam of these popular abstractions is shown as a class diagram in figure 1. During design studio sessions, the first author has observed that both novice and experienced software engineers quickly identify the abstraction concepts of "screen" and "film", but novice and some experienced software engineers struggle harder to delineate the concepts of "showing" and "booking". Experienced engineers often consider the possibility of a "ticket" abstraction. In figure 1, the "ticket" abstraction is conceptually subsumed by the "payment" abstraction.

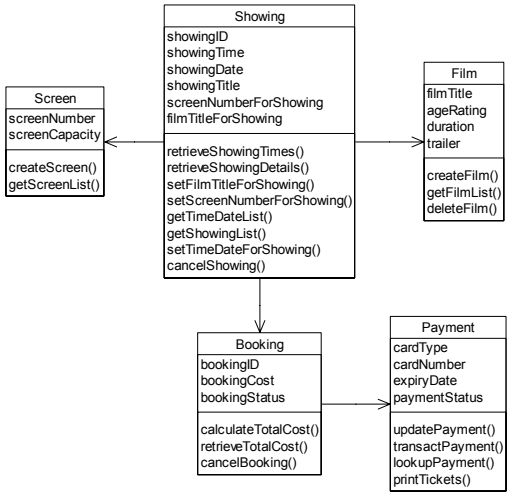

Figure 1: amalgam of human performed design

6.3 COM fitness of human-performed design

In order to establish a baseline cohesion fitness value for comparison with the EAs, the COM fitness of the human-performed design has been manually calculated using the sets of attributes, methods and uses specified above. The results of the manual calculation are presented in table 1.

Table 1. COM fitness of human performed design

Design concept (class)	COM fitness value
Screen	1.0
Showing	0.2185
Film	0.75
Booking	0.555
Payment	0.625

The COM cohesion fitness value for the human-performed design as a whole, expressed as the average of all individual design class COM values, was 0.62975.

7. EA design search results

Optimum settings for GA parameters have been chosen empirically when used with the software design search space representation and were as follows:

- population size 100
- number of generations 100
- crossover rate 70%
- mutation rate 3%

Optimum settings for EP parameters have also been chosen empirically as follows:

- population size 100
- number of generations 1000

Both the average COM fitness of every generation and the maximum COM fitness of any individual within the population of a generation have been recorded as the EA search for the cinema booking system case study evolved. For each algorithm, 10 evolutionary searches have been conducted, the population being initialized at random prior to each evolutionary search. At the end of the evolutionary search, "convergence" occurred if one or more individuals of superior fitness took over the entire population i.e. despite the variation pressure, fitter individuals no longer emerged during the remainder of the evolutionary search. The mean average COM fitness results of the 10 searches are shown in table 2, together with the average number of classes found in the final generation of the search, and an indication of the generation number at which convergence occurred.

Table 2. EA population average search results

	GA (Tournament)		GA (Roulette wheel)		EP	
	COM Fitness	2 SDs	COM Fitness	2 SDs	COM Fitness	2 SDs
Average population fitness for first generation	0.1096	0.008	0.1096	0.0125	0.1087	0.0066
Average population fitness for final generation	0.6436	0.094	0.6121	0.1294	0.6137	0.1447
Average number of classes in final generation	7.2	1.875	7.4	1.6895	4.1	1.4992
Generation at which convergence occurs	31.9	7.965	49.4	16.92	974.46	52.96

For GA, the average COM fitness for the final generation for both tournament and roulette wheel selection is similar, as is the average number of classes in the final generation. However, convergence to a local optimum is quicker with Tournament selection (generation 32) than roulette wheel (generation 49). Also, it has been observed that convergence with tournament selection resulted in a single individual taking over the entire population, whereas convergence with Roulette wheel involved different average fitness values repeating over a 3-5 generation cycle with no single individual taking over the population. Figure 2 shows the average and maximum fitness achieved as the population evolved for GA search. Both selection operators are shown for comparison.

Figure 2 also reveals that the average and maximum COM fitness of the GA population with roulette wheel selection lagged behind tournament in terms of generation number. This finding is consistent with authors such as Deb [17] (p89) who report that tournament selection has better or equivalent convergence properties when compared with other reproduction operators in the literature. This may be due to the fact that in tournament selection, for the next generation of the population to be created, each individual must take part in two tournaments i.e. there must be two tournament "passes" of the parent generation to create the offspring generation. Thus an individual of superior fitness is likely to generate two individuals in the offspring generation whereas an individual of inferior fitness is likely to generate none. This contrasts with the stochastic selection of the roulette wheel operator, where the probability that an individual will pass from the parent generation to the offspring depends upon its fitness proportion.

It seems likely that a further contributory factor to the rapid convergence of GA is the limitations of the trans-positional crossover (TPX) operator. There are two circumstances where TPX fails to exchange information between two individuals: (i) where the position of a method or attributes is the same in both individuals, and (ii) where the method or attribute is the only method or attribute in an individual. Such constraints are necessary to ensure the consistency of the sets of attributes and methods in an individual is not violated. One consequence, however, is that where classes of one attribute and one method existed with one use, the class achieved maximum COM fitness (1.0) and would not be further perturbed by TPX. Thus the high selection pressure of the GA, when combined with the weaker exploratory pressure of TPX, gave rise to quicker convergence when compared with EP.

For EP, the average population COM fitness in the final generation is similar to that achieved by the GA, both with tournament and roulette wheel selection. It is notable how similar the final average COM fitness values of the three EAs are, although the variance about the values increased from GA tournament, to GA roulette wheel, to EP. However, EP differed significantly from GA in terms of the average number of classes per individual in the final generation (4 for

EP as opposed to 7 for GA), and the generation at which convergence occurred (974 for EP, 49 for GA roulette wheel, 32 for GA tournament). The average and maximum fitness achieved for EP is shown in figure 3.

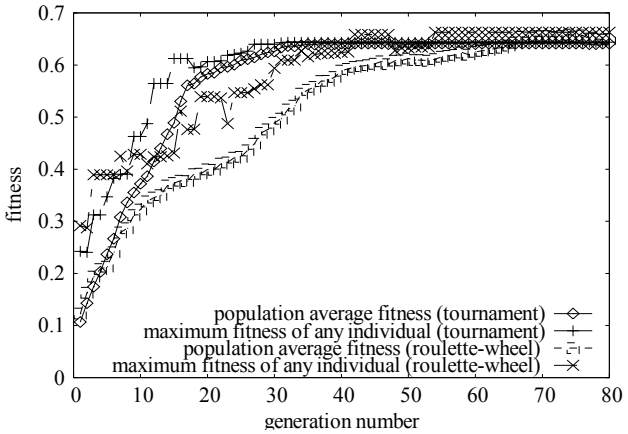

Figure 2. COM fitness for GA.

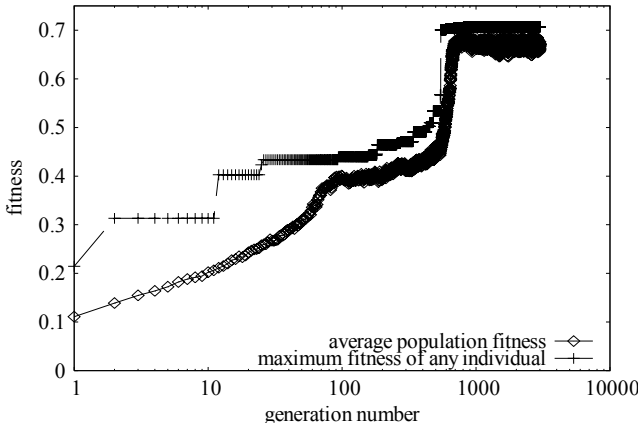

Figure 3. COM fitness for EP.

The higher number of generations required to achieve convergence and the finding that convergence involved average COM fitness values repeating over a 5-10 generation cycle with no single individual taking over the population suggest that the balance between exploitation and exploration pressures were different for EP when compared to the GA. In EP, selection of the next generation is achieved by a stochastic mechanism. There seems no logical reason to suggest that the selection pressure exerted by this mechanism should be greatly different from the selection pressure brought to bear by the roulette-wheel selection on the GA, which is also stochastic.

Additionally, reading the standard deviation values of average COM fitness in the final generation from table 2, a greater variety among individuals is evident in EP when compared to GA. (2 standard deviations from the mean reveals approximately 95% of COM fitness values). This may be explained by the greater exploration of the search space achieved by EP mutation. EP mutation is implemented by swapping both at class group level and attribute / method level within single individuals. Unlike TPX, there are no circumstances in which class level mutation fails to perturb an individual's attributes and methods, resulting in a more effective exploration of the search space. The greater exploration of the search space achieved by EP explains why the number of generations required to achieve convergence is greater in EP compared to GA.

It was realized at this point that mutation at both class and attribute / method levels is better suited to the search space representation insofar as mutation inherently respects the invariance of the sets of attributes and methods encapsulated in their classes. This contrasts with TPX, which violates class encapsulation. Thus EP mutation proves to be algorithmically simpler and more efficient at promoting variety than GA TPX.

A comparison of the average generation COM fitness achieved by the evolutionary searches of the three EAs is shown in figure 4.

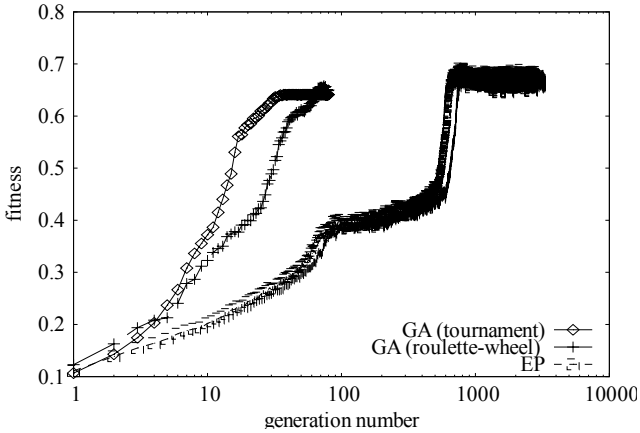

Figure 4. Average COM fitness for three EAs.

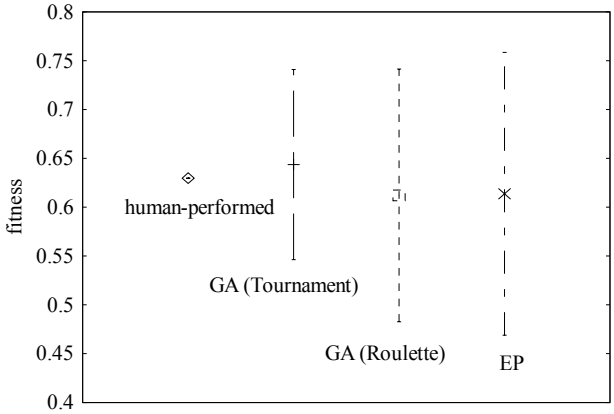

Figure 5. Fitness of human design and EA search.

To summarize, GA exhibits greater exploitation pressure through tournament selection. GA also exhibits an evolutionary balance of high exploitation and low exploration when compared to EP, resulting in fast convergence on a single individual design. Conversely, EP shows evidence of an exploitation pressure comparable to the roulette-wheel selection of GA. Significantly, however, EP exhibits a high exploration pressure when compared with GA, resulting in a slower convergence with no single individual taking over the population. In short, while the stochastic selection operators of both GA and EP appear to produce a comparable selection pressure, the variation pressure of EP is greater than GA due to the effectiveness of the mutation operator.

Lastly, for visual comparison, figure 5 shows the COM cohesion value for the human-performed design with the average COM cohesion values for the EAs (as given in table 2). In terms of COM cohesion, the EAs produced conceptual software designs of similar cohesion to human performance.

8. CONCLUSIONS

The paper set out to compare genetic algorithms and evolutionary programming when applied to conceptual software engineering design. It is encouraging to note that the COM metric results produced by the two evolutionary algorithms (GA and EP) achieves a close similarity with the human-performed design. A greater variety among designs is evident in EP compared to GA.

We conclude that it is highly unlikely that the large number and great diversity of optimal conceptual software design variants produced by the EAs could have been achieved by a human software engineer manually, and so suggest that a search based approach using EAs has significant potential when integrated in computationally intelligent tools for conceptual software engineering design.

It is interesting to speculate how the characteristics of the EAs could be harnessed by the software engineer, given that many conceptual designs of roughly equivalent COM optimality are evident for a given design problem. Perhaps when time is of the essence and choice of optimal conceptual design is less crucial, the speed with which GA with tournament selection produces an optimal design is attractive. On the other hand, if variety of conceptual designs is important, particularly should the software engineer wish to interact with the on-going evolution, EP may present an attractive option. Thus we also conclude that human software engineer interaction will likely be an essential component of any future successful computational tool support of the software engineering conceptual design process.

9. References

[1] Guindon, R. Designing the design process: exploiting opportunistic thoughts. *Hum.-Comput. Interact., 5*, 2-3 (1990), 305-344.

[2] Nandhakumar, J, and Avison, D. E. The fiction of methodological development: a field study of information systems development. *Inf. Technol. Peop., 12*, 2 (1999), 176-191.

[3] Glass, R. L. *Facts and fallacies of software engineering*. Addison-Wesley, 2003.

[4] Simons, C. L., Parmee I. C., and Coward, D. P. 35 years on: to what extent has software engineering design achieved its goals? *IEE Proc. – Softw., 150*, 6 (December 2003), 337-350.

[5] Parmee, I. C. *Evolutionary and adaptive computing in engineering design*. Springer-Verlag, 2001.

[6] Parmee, I. C. Improving problem definition through interactive evolutionary computation. *Artif. Intell. Eng. Des. Anal. Manuf.*, *16*, 3 (June 2002), 185-202.

[7] Parmee, I. C. Human-centric intelligent systems for exploration and knowledge discovery. *Analyst, 130*, (2005), 29-34.

[8] Simons, C. L., Parmee, I. C. Defining the search space for conceptual software designs. Technical Report TR011105, available online: http://www.cems.uwe.ac.uk/~clsimons.

[9] Harman, M. and Jones, B. Search-based software engineering. *Inf. Softw. Technol., 43*, 14 (Dec. 2001), 833-839.

[10] Lee, J. Introduction to software engineering with computational intelligence. *Inf. Softw. Technol., 45*, 7 (May 2003), 371-372.

[11] Seng, O., Bauer, M., Beil, M., and Pache, G. Search-based improvement of subsystem decompositions. In *Proceedings of the Genetic and Evolutionary Computing Conference 2005 (GECCO '05)* (Washington, DC, USA, 25-29 June 2005), ACM Press, 2005, 1045-1051.

[12] Gokhale, S. S. Software application design based on architecture, reliability and cost. In *Proceedings of the 9th International Symposium on Computers and Communications (ISCC 2004) (Alexandria, Egypt,* June 28 – July 1) 2004, 1098-1103.

[13] Lo, S.-C., and Chang, J.-H. Application of clustering techniques to software component architecture design. *Int. J. Softw. Eng. Knowledge Eng., 14*, 4 (Aug 2004), 429-439.

[14] Jacobson, I., Christerson, M., Jonsson P. and Overgaard, G. *Object-oriented software engineering: a use case driven approach,* Addison-Wesley, 1992.

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

[16] Goldberg, D. E. *Genetic algorithms for search, optimization, and machine learning.* Addison-Wesley, 1989.

[17] Deb, K. *Multi-objective optimization using evolutionary algorithms.* John Wiley & Sons, 2001.

[18] Fogel, L. J., Owens, A. J. and Walsh, M. J. *Artificial Intelligence through simulated evolution.* John Wiley & Sons, 1966.

[19] Back, T. *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.

[20] Eiben A. E. and Smith, J. E. *Introduction to Evolutionary Computing.* Springer-Verlag, 2003.