The Journal of Systems and Software 103 (2015) 440–451

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

The influence of search components and problem characteristics in early life cycle class modelling

Jim Smith*, Chris Simons

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

ARTICLE INFO

Article history:
Received 1 December 2013
Revised 23 October 2014
Accepted 15 November 2014
Available online 29 November 2014

Keywords:
Search-based software engineering
Class modelling
Meta-heuristics

ABSTRACT

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

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

1. Introduction

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

Our contention is that because of its complex, subjective nature, class modelling should be tackled via interactive search, augmented with a surrogate fitness function to prevent user fatigue. Therefore ideally the choice of search method should consider the ease with which it can support users' input via actions such as “freezing” satisfactory parts of designs. Previously (Simons and Smith, 2012, 2013) we compared ACO and EAs as global search algorithms for this task, concluding that performance issues aside, there are practical reasons for preferring to use some heuristics. For example, in an ACO “freezing” of partial solutions can be simply achieved via direct changes to the pheromone table. In contrast, it would necessitate manipulation of an EA's recombination and mutation operators on-the-fly, although progress has been made in this direction by interleaving phases of human manipulation and evolution (Vathsavayi et al., 2013).

However, these considerations are orthogonal to the comparative search performance, so understanding the latter is vital before we can disregard it, and clearly algorithmic simplicity cannot be a substitute for poor performance. Initial investigations (Simons and Smith, 2013) showed that in their canonical form EAs outperformed ACO. In a

* Corresponding author. Tel.: +44 117 3283161.
E-mail address: james.smith@uwe.ac.uk (J. Smith).
URL: http://www.bit.uwe.ac.uk/~jsmith (J. Smith),
http://www.bit.uwe.ac.uk/~clsimons (C. Simons)

http://dx.doi.org/10.1016/j.jss.2014.11.034
0164-1212/© 2014 Elsevier Inc. All rights reserved.

recent paper we extended this work to incorporate the effect of local search to create two different examples of the class of memetic algorithms (MAs) (Krasnogor and Smith, 2005). Those preliminary studies still showed that the MAs using evolution as the global search component (hereafter M-EAs) found higher quality solutions than those based on ACOs (hereafter M-ACO). It should be noted that for the sake of "fairness" the ACOs in those papers did not make any use of heuristic or instance-specific information.

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

2. Background

2.1. What makes a class modelling problem hard?

Early lifecycle class modelling is an intensely human activity wherein relevant concepts and information relating to a design problem are identified. High quality class models are crucial as the basis of subsequent software development activities, as inferior designs can lead to deleterious and costly down-stream consequences. Starting from use case descriptions or user stories from the design problem domain, various required software system actions and data are identified. In the object-oriented paradigm such actions and data correspond to candidate 'methods' and 'attributes' to be grouped by means of the 'class' construct. Class models thus reveal how these groupings relate to relevant concepts and information in the problem domain. There is evidence to suggest that act of early lifecycle class modelling is non-trivial and demanding to perform, not least due to the scale and complexity of the problem domain. For many problems the number of methods and attributes can run to hundreds, with a corresponding multiplicity of classes. Petre (2009) has suggested that software design problems are often wicked: too big, too ill-defined, too complex for easy comprehension and solution. Sometimes the problems are only fully understood after they are solved. Solving such problems is rarely a matter of brute force or routine. Glass (2003) goes further, suggesting that the scale and complexities of some software designs may be beyond human comprehension. There is also evidence that designers are blessed with varying degrees of modelling talent. Even for experienced modellers, Glass notes that designer performance may vary from 28:1 from the best to the worst. Curtis et al. (1998) also note a range in talent, observing that only super-designers can reason across the full breadth and depth of complex, ill-structured problems in order to fully consider the consequences and decisions of modelling decisions. From the field of education there is evidence that class modelling is difficult to learn. In a study of 740 undergraduates and 135 design problems, Svetinovic et al. (2005) observe that with respect to concept identification, *"some students just don't get it"*.

To help overcome these difficulties, manual heuristics and search strategies are available. For example, Larman (2008) suggests that the required actions of the software system-to-be be regarded as responsibilities, i.e., a contract or obligation of a class-to-be. Class modelling then becomes an exercise in assigning responsibilities to candidate classes. Larman proposes General Responsibility Assignment Software Patterns (GRASP) to guide the class modeller. GRASP reflect modelling principles such as separation of concerns, high internal class cohesion, and low coupling between classes. Wirfs-Brock and McKean (2003) describe a responsibility-driven design approach to class modelling as a process of discovery and invention. They propose a manual search strategy, in which modellers make educated guesses about the kinds of inventions needed based on the nature of the problem domain and the things that are critical to it. In this manual search candidate models are evaluated from the perspectives of: information flowing through the model; decision making, control and coordination activities; and representations of real-world things that the model needs to know about.

2.2. Search-based software engineering

Search-based software engineering (SBSE) is a well-established discipline, applied across the whole software development lifecycle (Zhang, 2014). Historically, comparatively little focus has been directed to the upstream stages, although this is beginning to be addressed. Typically metrics relating to coupling and cohesion are used to guide meta-heuristic search of design spaces of object-oriented class models. Bowman et al. (2010) used a multi-objective EA to optimise designs for a number of pre-specified metrics, but only considered a single problem instance. Simons et al. (2010) and Simons and Parmee (2012) applied interactive EAs, using linear regression to learn a surrogate fitness model combining coupling with a number of "elegance metrics" to approximate users' subjective preferences for different problems. Working slightly later in the development lifecycle, Sievi-Korte et al. (2010) used an M-EA, and Vathsavayi et al. (2013) interleaved human and evolutionary adaptation of the usage of patterns.

Although any search algorithm could be used in SBSE, research effort has tended to concentrate on EAs. Previously we compared the use of ACOs and EAs for this problem (Simons and Smith, 2012, 2013), concluding that given sufficient computational budget, global search via EAs was more effective at finding high quality solutions than that using ACO. When the computational budget was reduced (as is, for example, often the case in interactive search) the situation was reversed. However, with both algorithms, and both representations examined, a major issue was dealing with the constraint that a valid class model should contain at least one attribute and at least one method. Those papers used penalty functions (all invalid models were given zero fitness) and the use of random regeneration of invalid solutions. Dealing more efficiently with this constraint would necessitate either a significant adaptation of the underlying global search heuristics, or the provision of a "repair" mechanism.

A well designed local search algorithm that systematically examines the effect of moving elements between classes provides a simple way of providing the latter, and also of improving valid solutions. In a recent paper we reported preliminary studies on the influence of such a component (Smith and Simons, 2013). Results showed that in the absence of other forms of information the M-EAs still discovered higher quality solutions that M-ACO, albeit over a longer timescale. This could be interpreted in one of two ways—either that the M-ACO are better given a limited computational budget, or that the M-EAs are better able to escape local optima. However, that paper did not permit any use of heuristic functions or other problem-specific adaptations. While such knowledge might require substantial alterations of operators within an evolutionary framework, it is standard practice in ACO research and is readily incorporated, as will be shown in later sections.