Search Based Approaches to Component Selection and Prioritization for the Next Release Problem

Paul Baker
Motorola Labs,
Viables Estate,
Basingstoke, UK.

Mark Harman
King's College
Strand, London
WC2R 2LS, UK.

Kathleen Steinhöfel
King's College
Strand, London
WC2R 2LS, UK.

Alexandros Skaliotis
King's College
Strand, London
WC2R 2LS, UK.

Keywords: next release problem, search based software engineering

Abstract

This paper addresses the problem of determining the next set of releases in the course of software evolution. It formulates both ranking and selection of candidate software components as a series of feature subset selection problems to which search based software engineering can be applied. The approach is automated using greedy and simulated annealing algorithms and evaluated using a set of software components from the component base of a large telecommunications organisation. The results are compared to those obtained by a panel of (human) experts. The results show that the two automated approaches convincingly outperform the expert judgment approach.

1 Introduction

Software development is becoming less and less associated with the development, *ab initio*, of a single software system and more and more an evolutionary process in which a system is incrementally developed over a series of releases. There is also an increase in the use of component based approaches, in which the next release in the evolutionary process is defined in terms of a set of additional components that augment the existing system to meet a set of constraints. These components may be pre-existing or planned. Systems often contain an integrated mixture of pre-existing components and newly built components.

Much has been written about the challenges of component based software engineering [5, 9], but the focus of this work has been on the problems of integration of components, testing, defining, delineating and assuring the interfaces between components and the verification of their intended behaviour and interactions.

However, the increasing use of component–led approaches to software evolution also presents higher level management choices. These decisions are typically made in the planing stage of the process and, as such, their influence can be far reaching; a poor judgment at this point in the development process can be extremely costly. Furthermore, because of the nature of most systems' evolution [11], today's decisions can have a dramatic effect upon the future evolution of the system.

A generic instantiation of this problem finds the manager considering several candidate components for which the data available include estimates of the cost of acquisition (third party purchase or in-house development), customer desirability, development time and expected revenue. The manager may also have information about the dependencies between components and may wish to include other factors in the decision making process, such as the priority given to each of the customers.

From the set of all components, the manager must search for a subset that balances these competing concerns in the best way possible. The manager may also want to rank (or prioritize) the components in some way based upon these trade-offs. For systems with more than a few simple components the search space is unmanageably large and complex, with the consequence that no manager can be expected to find optimal choices that balance the constraints without some form of automated support. This is the 'Component Selection Problem'. A closely related problem is the 'Component Prioritization Problem', in which the manager seeks to order the components under consideration into a priority ranking. The Component Selection Problem helps the manager decide which component combinations will make sensible choices for future evolution, while the Component Prioritization Problem helps with planning the order in which components should be tackled first when planning the next release.

This paper presents the results of study of the application of automated approaches to the solution of these two related problems. The paper formulates both problems in terms of a series of feature subset selection problems and presents algorithms for their solution using search based software engineering.

The paper evaluates the approach using real world data from a large global telecommunications company, concerned with selection and prioritization of forty candidate software components for mobile telecommunications devices. The real world data has been anonymized and domain specific information has been removed to protect confidentiality. However, no data values have been changed and so all results reported are real, replicatable and unaffected by this anonymisation process.

The experimental study compares the results obtained from greedy and simulated annealing algorithms with the ranking produced by expert judgment. The principal contributions of this paper are as follows:

1. The paper presents results which illustrate the application of greedy algorithms and simulated annealing to software selection and ranking problems using real world data from a large telecommunications organisation. The results are also compared with an expert-ranking, showing that the automated search outperforms the expert ranking.

2. The results reveal that the simulated annealing approach convincingly outperforms the greedy approach, and that the data set contains combinations of features that cause the known sub–optimal behaviour of greedy approaches to manifest themselves.

3. As well as discriminating between greedy and simulated annealing approaches, the study also investigates the commonality between the solutions found, showing how the use of search based approaches in this problem domain can be used to shed light on the reasons for the differences in rankings produced. The results from the study reveal that the difference between the best rankings found and the worst can be explained by the inclusion of a few very costly features. In this way the approach allows the experts to pay particular attention to 'controversial choices' (those not common to solutions found by all techniques).

4. The results for the simulated annealing algorithm were found to be robust over many different executions of the algorithm with identical parameter settings and also over different runs with different parameter settings, indicating that the simulated annealing approach is robust for this problem and data set.

The rest of the paper is organised as follows: Section 2 states the research problems investigated in this paper more rigorously, while Section 3 presents a brief overview of the search algorithms studied. Section 4 describes the experimental methods and Section 5 presents the results of the experiments and discusses the findings. Section 6 briefly describes related work and Section 7 concludes.

2 Problem Statement

In this paper we are working with a model of software development in which software systems are constructed as an agglomeration of sets of components with well defined behaviours and interfaces. This model of software development is becoming more prevalent with the advent of component–based software engineering and with the increasing reliance of large organisations upon out–sourcing of software development, service–based approaches and the construction of architectures into which trusted and semi-trusted components are assembled.

The results reported in the paper come from the application of our approach to software component selection for a large telecommunications organisation that has a range of portable communications devices, each of which must contain features that are attractive to the users and important to the company in terms of their revenue–to–cost ratio.

In the application domain reported upon here, these components are independent 'add-ons', to the base system, i.e., these components will determine special, additional features or functions of the next release in the evolution of the software. Each component is characterised by a set of values, including estimates of the cost of acquisition, customer desirability and expected development time and revenue. We associate a weight (or score) with each component where we combine the cost of acquisition and development time to a single cost value c_i, and customer desirability and expected revenue to a weight value w_i, and the value of the item, x_i where i is an index of the components.

In the feature or component selection problem, we are given a set of such components and want to determine a subset that maximises the total sum of weights while minimizing the total cost of the selected components.

This problem, like most realistic optimisation problems, requires the simultaneous optimisation of more than one objective function. Similar to the traditional portfolio optimisation problem that attempts to simultaneously minimize the risk and maximise the fiscal return, we need to find some trade-off between the criteria to ensure a satisfactory design.

Such multiobjective problems can be solved by combining the multiple objectives into one scalar objective. Another approach is to bound one criteria while optimising the other. This yields solutions that are optimum and nondominated; there are no other solutions superior in all objectives. These, so–called Pareto optimal solutions, plot a Pareto optimum curve that gives the best possible insight into trade-off solutions.

By bounding the total sum of costs to K, we can formulate the component selection problem for a particular given bound of total cost as an optimisation 0-1 knapsack problem:

$$\begin{array}{lll} \text{maximise} & \sum_{j=1}^{n} & w_j x_j \\ \text{subject to} & \sum_{j=1}^{n} & c_j x_j \leq K, \qquad x_j \in \{0,1\}. \end{array}$$

The knapsack problem is known to be NP-hard. However, it can be solved by a pseudo-polynomial algorithm using dynamic programming [13]. The algorithm runs in $O(n^2 w)$ time (where n is the number of components) and therefore depends on the optimum value for w that can be found within K. One can transform this algorithm into a polynomial-time approximation scheme (PTAS) by truncating the last t decimal digits of all w_i.

However, for this application such truncation may not be desirable; choices of components can have a significant impact on the organisation's profitability, so we are interested in solutions that are as close as possible to the Pareto optimum. Therefore, we seek to solve each of the set of individual knapsack problems governed by each possible budget choice. We compare the performance of each search with a greedy approach and we also compare the implied ranking from the set of feature subset selection problems with the pre-existing expert ranking.

3 Algorithms Studied

This section describes the two automated techniques for solving the sequence of feature subset selection problems in more detail.

3.1 Simulated Annealing

Simulated annealing algorithms act within a search space in accordance with a certain neighbourhood structure or a set of transition rules, where the particular steps are controlled by the value of an objective function. In this case, the search space is the set of feasible solutions for a given set of n features and a given budget limit K. We shall denote this space by $\mathcal{F}$. Clearly, each component is included in the solution at most once. Therefore, we can represent a solution S as a vector $\{x_1, \ldots, x_n\}$, where $x_i \in \{0, 1\}$. Thus the size of $\mathcal{F}$ can be upper bounded by $|\mathcal{F}| \leq 2^n$, because it consists of the set of all subsets of n features. Such a space is too large for enumeration of all possible solutions, suggesting that a search based approach may be appropriate.

To describe the neighbourhood of a solution $S \in \mathcal{F}$, we define a **neighbourhood function** $\eta(S) : \mathcal{F} \rightarrow \wp(\mathcal{F})$. The neighbourhood of S is given by $\eta(S) \subseteq \mathcal{F}$, and each solution in $\eta(S)$ is called a neighbour of S. Our neighbourhood transition is a flip of a single variable in the solution vector. Thus, a neighbour $S' \in \eta(S)$ of a solution $S = \{x_1, \ldots, x_n\}$ is a vector $\{x'_1, \ldots, x'_n\}$ where exactly one $x_i \neq x'_i$ and $\sum_{j=1}^{n} c_i x_i \leq K$.

The **objective** is to maximise the total score of feasible subsets, where 'feasible' means the total cost of selected components is less or equal to the budget. Hence, we define the objective (or 'fitness') function, $\mathcal{Z}$ as follows:

$$\mathcal{Z}(S) \quad := \quad \sum_{j=1}^{n} w_j s_j, \tag{1}$$

where $s_j = 1$ iff the j^{th} component has been selected and therefore, is a member of the solution subset. Furthermore, we set

$$\mathcal{F}_{\max} \quad := \quad \big\{\, S \mid S \in \mathcal{F} \text{ and } \forall S' \big(S' \in \mathcal{F} \rightarrow \mathcal{Z}(S') \leq \mathcal{Z}(S)\big) \,\big\}.$$

In simulated annealing, the transitions between neighbouring elements depend on the objective function $\mathcal{Z}$. Given a pair of feasible solutions $[S,\, S'],\, S' \in \eta(S)$, we denote by $G[S,\, S']$ the probability of generating S' from S and by $A[S,\, S']$ the probability of accepting S' once it has been generated from S. Since we consider a single step of transitions, the value of $G[S,\, S']$ depends on the set $\eta(S)$. In most cases, a uniform **generation probability** with respect to S is taken, given by

$$G[S,\, S'] \quad := \quad \frac{1}{|\,\eta(S)\,|}.$$

The **acceptance probabilities** $A[S,\, S'],\, S' \in \eta(S) \subseteq \mathcal{F}$ are derived from the underlying analogy to thermodynamic systems and are the following:

$$A[S, S'] := \begin{cases} 1, & \text{if } \mathcal{Z}(S) - \mathcal{Z}(S') \leq 0, \\ e^{-\frac{\mathcal{Z}(S) - \mathcal{Z}(S')}{c}}, & \text{otherwise,} \end{cases}$$

where c is a control parameter having the interpretation of a *temperature* in annealing procedures. The actual decision as to whether or not S' should be accepted for $\mathcal{Z}(S') < \mathcal{Z}(S)$, is performed in the following way: S' is accepted, if

$$e^{-\frac{\mathcal{Z}(S) - \mathcal{Z}(S')}{c}} \;\geq\; \rho,$$

where $\rho \in [0, 1]$ is a uniformly distributed random number. The value ρ is generated in each trial if $\mathcal{Z}(S') < \mathcal{Z}(S)$.

Finally, the probability of performing the transition between S and S', $S, S' \in \mathcal{F}$, is defined by

$$\mathbf{Pr}\{S \rightarrow S'\} = \begin{cases} G[S, S'] \cdot A[S, S'], & \text{if } S' \neq S, \\ 1 - \sum\limits_{Q \neq S} G[S, Q] \cdot A[S, Q], & \text{if } S' = S \end{cases}$$

By definition, the probability $\mathbf{Pr}\{S \rightarrow S'\}$ depends on the control parameter c.

For the **cooling schedule** we follow the analysis presented in [18]. The starting "temperature" $c(0)$ is defined by

$$e^{-\frac{\Delta \mathcal{Z}^{\max}}{c(0)}} \quad = \quad 1 - p_1, \qquad c(0) = -\frac{\Delta \mathcal{Z}^{\max}}{\ln(1 - p_1)},$$

where p_1 is a small positive value.

The decremental rule of the cooling schedule is given by the simple relation

$$c(t+1) := (1 - p_2) \cdot c(t), \tag{2}$$

where p_2 is a small value larger than zero. The stopping criterion is reached when the temperature falls below a value $c(t_{fin})$ which is chosen to be close to zero (for exact values of parameters, see Section 4).

3.2 Greedy Algorithm

For the greedy algorithm, all components have been sorted according to their weight value w_i. Then all components with the highest weight are included in the solution until the budget bound has been reached. For the case where the inclusion of the next highest-scoring feature exceeds the budget, our procedure checks whether one of the following components can still be fit into the budget.

The sorted components are stored in an array `cost` such that the cost of the component with the highest score is stored at `cost[1]`. The following pseudocode describes how our greedy algorithm selects the components. A value of 1 in `solution[i]` indicates that the i^{th} highest-scoring component is included in the solution.

```
for(i = 1 to number_of_components)
   solution[i] = 0;
for(i = 1 to number_of_components)
   if(current_cost + cost[i] <= budget) {
      current_cost += cost[i];
      solution[i] = 1;}
```

4 Experimental Methodology

For this experimental study, we used a component base containing forty software components, all of which are candidates to be included in possible future evolution of the software controlling a mobile telecommunications device. The set of components were ordered according to an expert ranking that denotes the importance of a component as judged by a panel of human experts from the company.

Currently, the component selection process would involve hand-picking as many high-ranked components as possible until a budget bound is met. As described in Section 3, we used a greedy algorithm and a simulated annealing-based procedure to investigate the behaviour of these two automated alternative methods to component selection and ranking.

Following a discussion with the human experts, it was decided that the first five entries of the list consisted of important components that should need to be present no matter how the device software might evolve. Therefore, these 'base components' were removed from the set of candidates and were not taken into consideration in the experiments we conducted. The remaining 35 components were the subject of the ordering experiment, using a series of 35 feature subset selection experiments for each ranking method. Each subset selection experiment, E_i was allocated a budget b_i; the budget implicitly required by the expert judgment that determines that the first i elements of the expert ranking should be selected as the set of features should only i features be included in a particular product.

In this way, the ranking problem is reduced to a set of feature subset selection problems for each automated algorithm applied. In each feature subset selection experiment, we recorded the overall score and the number of features that we were able to include. The results from the experiments are presented in the next section.

Both approaches were implemented in C++ and optimised for fast execution. The entire set of experiments to produce a ranking using the greedy approach took practically no noticeable time on standard computing equipment, while the simulated annealing procedure took about 30 seconds on average. The parameter setting for our simulated annealing procedure has been derived following the analysis by Steinhöfel et al. [18] and are as follows: $p_1 = 0.8$, $p_2 = 0.2$, $L = 15000$ and $c(t_{fin}) = 0.005$, where L denotes the number of steps at each level of the temperature.

The greedy algorithm is entirely deterministic and so only one experiment was performed for each of the 35 feature subset selection problems. However, the simulated annealing algorithm, in common

with other meta-heuristic search techniques commonly used in Search Based Software Engineering (SBSE) [7, 8], exhibits a certain degree of randomness. To control for this randomness, multiple runs of each feature subset selection problem were executed and the results for each were compared.

Strikingly, we found that the results for all of these multiple runs were identical; only the execution time changed. This observation remained, even when the parameters of the simulated annealing algorithm were changed. This finding tends to suggest that the results obtained are highly robust and are not unduly affected by either the inherent randomness present in the approach, nor by any unwanted sensitivity to algorithmic parameter settings.

5 Results and Analysis

A summary of the results can be found in Figure 4. The column headings of the form C_X refer to costs, Z_X refers to score (fitness) obtained and S_X refers to the size of the set of features selected (the more the better). The subscripts to these three labels indicate the selection/ranking method used ('GR' denotes GReedy algorithm, 'ER' = denotes (human) Expert Rank judgement and 'SA' denotes Simulated Annealing). The final two columns report the size of the intersection of features from two sets. That is, they record the number of features that are common between two solutions to the selection problem.

The performance of the three approaches to ranking features and selecting sets of features for a given budget, can be better understood by visualisation. Figures are used to illustrate the performance of the algorithms in terms of score and feature set size and the behaviour of the algorithms in terms of commonality of feature selection.

Figure 1 presents the overall score of each method for the thirty-five budget bounds. The initial observation is that the simulated annealing procedure yields the best score in every experiment, with the greedy algorithm following and the expert judgement approach performing least well.

Clearly, the graph of these three performance functions must (naturally) meet at the last data point, because this limiting budget permits the inclusion of all the components. The graph suggests that both simulated annealing and the greedy algorithm produce considerably better sets of components than those selected by expert judgement. For example, with a budget limit of 1960, $\mathcal{Z}_{GR}$(the score for the greedy algorithm) is about 136% better than $\mathcal{Z}_{ER}$(the score for the expert ranking) and $\mathcal{Z}_{SA}$(the score for the simulated annealing algorithm) is even better with a 158% increase. Even for higher budgets, the increase is still appreciable. For instance, with a budget of 5670 we were able to get $\mathcal{Z}_{SA}$ to be 27% better than $\mathcal{Z}_{ER}$.

The substantial gains in score can be explained by looking at the number of components that each solu-

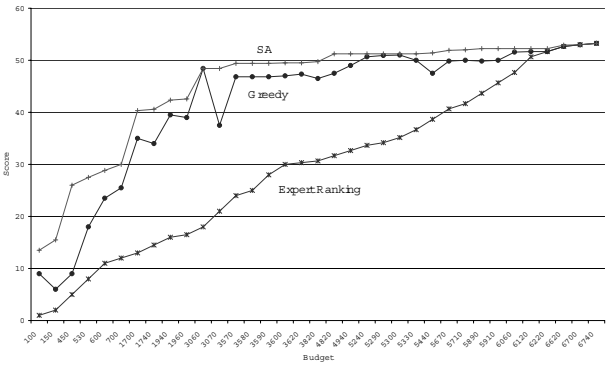

Figure 1: Scores achieved by each method for each budget bound

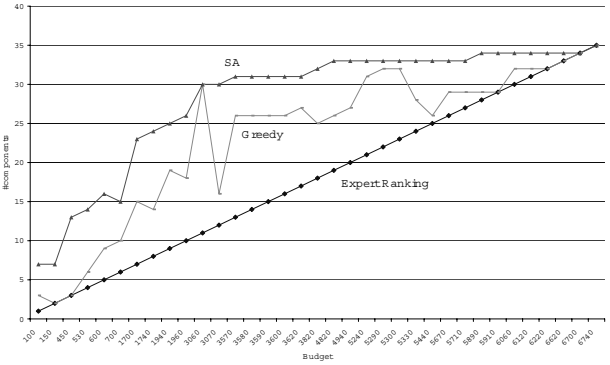

Figure 2: Number of components selected by each method for each budget bound

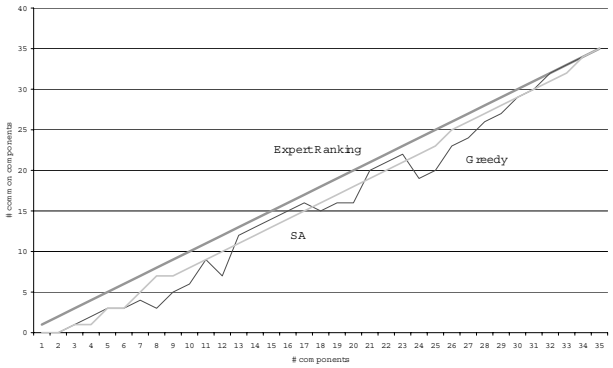

Figure 3: Common components between the three methods and the expert ranking approach

tion contains. Figure 2 shows the number of components that each approach was able to include for each budget limit. By construction of the experiment, the number of components of the expert-ranked solutions increases by one with each increased budget, providing a baseline for comparison.

The figure indicates that the two automated methods manage to include many more features in each run, thereby obtaining higher fitness scores. These findings suggest that some components with a relatively low score and relatively high cost are ranked 'artificially' highly by the human experts.

In most cases, the dramatic difference in score is due to the experts including one or two high-cost components. For example, with a budget of 3570, the expert-ranking method would result in a set of 13 components, with 12 of these also included in the greedy algorithm's solution (see Figure 4). However, by omitting just one human-selected component, the greedy algorithm manages to include a further 14 components, resulting in a dramatic increase in score. For the same budget, the simulated annealing procedure omits two components from the experts' choice, allowing it to include a further 19 components.

Figure 3 illustrates the number of common components selected for each budget. The figure shows the number of components that are common to the solutions found by the simulated annealing algorithm and the expert judgement approach and the number common to the greedy algorithm and the expert judgement approach.

The x-axis represents the thirty-five runs. The line $x = y$ is included to provide an indication to how close each of the other two lines come to the maximum possible level of commonality. The simulated annealing procedure manages to include a large percentage of the components preferred by the expert in every run except in the first few. In fact, except for the first four runs, the algorithm omits a rounded-up average of only two components per run. It is also important to note from the relationship between components chosen by each search technique (represented by the results in Figure 3) that the simulated annealing approach has a closer agreement to the human ranking, despite producing better results than the greedy approach. This is important, because the expert ranking drawn up by the human–based approach is likely to take into account, unstated, implicit factors which have not and cannot have been factoring into the assessment of fitness.

The users of any automated technique for producing ranking results are unlikely to want to deviate enormously from the expert judgement ranking, even if the apparent benefits are high because of the influence of these unstated assumptions, which can only be taken into account by the experts. Of course, a set of results such as these, form a natural starting point for a discussion between the experts and the experimenters and, indeed, this is precisely what has happened in practice in our experience.

The benefits of using the two automated methods over the expert ranking are obvious. Furthermore, simulated annealing convincingly outperforms the greedy algorithm; it performs better than or equally well as the other methods in all experiments. Although the scores and number of features in each case clearly support the above statement, there is another fundamental reason that would make simulated annealing a more suitable approach.

It would be natural to suppose that the higher the budget is, the higher the overall score would be. However, as is well known, greedy algorithms do not necessarily observe this 'natural intuition'. For our data set, this can be observed for budget limits 3070 and 3570 (see Figures 1 and 4). The large drops in score and number of components for the greedy algorithm

are due to the way the algorithm works (see Section 3).

The algorithm starts from the highest-scoring components, considering each in turn, progressing towards the lowest-scoring components until no unused component can be added without surpassing the budget bound. In case the component's cost does not allow it to be selected, the greedy algorithm ignores it and proceeds with the next one. For example, at budget B_i, component C_x may not be included because of its cost, but components C_{x+1} and C_{x+2} may be selected because $c_x > c_{x+1} + c_{x+2}$. In the next iteration, at budget B_{i+1}, C_x is included and there is no budget room available for any other components. A 'drop' in score over two iterations will be noticed if $w_x < w_{x+1} + w_{x+2}$.

What these data reveal is that the search space for this component selection and ranking problem is not well suited to a greedy approach, although it is interesting to note that the greedy approach is still able to surpass the score and number of features available using the expert judgement approach.

6 Related Work

General ordering and feature subset selection problems have been widely studied in the operations research, meta-heuristic search and evolutionary computation communities [13]. The work reported in the present paper on feature subset selection and ordering (ranking) follows this general pattern, but is most closely related to the previous work on ordering and selection problems in Software Engineering, in in particular to previous work on Search Based Software Engineering (SBSE). This previous work has concerned regression testing (selection and prioritization), project planning (ordering and selection) and requirements analysis(selection).

Much of the work on test case prioritization [14, 17, 19] has concerned the application of greedy algorithms. However, Nashat Mansour [12] empirically compared five regression test selection algorithms: Simulated Annealing, Reduction, Slicing, Dataflow and Firewall algorithms. However, unlike the programs studied by Rothermel and his colleagues, the programs used by Mansour were small laboratory programs. Wong et al. [19] presented a technique that combines test set minimization and prioritization to select test cases, according to the criterion of 'increasing cost per additional coverage'.

Other examples of selection and ordering problems in Search Based Software Engineering come from work on project planning [1, 2, 6] and the Next Release Problem in requirements engineering.

Antoniol et al. [1, 2] address the problem of ordering the work packages of a massive maintenance project using a combination of search techniques and a queueing simulation. A variety of encodings and search algorithms are used. The primary difference between the work of Antoniol et al. and that presented here is that the fitness function employed by Antoniol et al. uses a simulation–based approach and that the items being ordered are work packages (all of which have to be completed) rather than components (only some of which may be used).

Chicano and Alba [6] also use search techniques for the software project planning problem. In their approach, the problem is formulated as a multi objective search problem in which each objective is combined into a single fitness function by means of weights applied to each sub-fitness function.

Kirsopp and Shepperd [10] also used search to address problems of project planning. In this case, the problem was cost estimation using a case–based reasoning approach [15, 16]. The estimates of features (such as cost) for a newly proposed project are calculated in terms of those historical projects found to be most similar to the newly proposed project. The search problem consists of determining the set of project features that forms a good basis for this predictive analogy. Kirsopp and Shepperd present results for a hill-climbing solution to the search for such a feature subset selection problem, based on real–world data for projects with 43 features. They also use their results to present visualisations of the search landscape.

One of the first papers that presented a solution to a software engineering problem as a feature subset selection search was the work on the 'Next Release Problem' by Bagnall et al. [3]. In this work, the problem was to determine the set of requirements that

should be included in the next release of a software system. Bagnall et al. formulated this problem as a feature subset selection problem and presented results from the application of several search techniques to synthetic data on hypothesized projects and their requirements sets.

We presented a poster at GECCO 2006 [4], which outlined our approach to ranking as feature subset selection, but this two-page poster contained neither detail nor results. The present paper is the first to apply search based software engineering techniques to a real-world next release problem and to demonstrate that the search based approach is able to out-perform expert judgement. The present paper is also the first to study robustness and commonality of features selected in the different approaches.

7 Conclusion and Future work

This paper has shown how ideas from search based software engineering can be applied to problems of software component selection and ranking, producing results which exceed those produced by expert judgement alone.

The paper reports on the application of both greedy and simulated annealing algorithms to this selection and ranking problem. The results were compared to expert judgement for a large-scale real–world set of features, representing a set of components in a software component base.

The results show that both of the automated techniques outperform the expert ranking, with the simulated annealing approach providing superior results to the greedy approach. The results also show a striking robustness of solution for the simulated annealing approach and a large degree of commonality between solutions found by the approaches (despite large differences in fitness achieved). In this way, it is possible to use the search approach to provide insight into the nature of the search problem and the characteristics of the set of data under study.

Future work will consider improved algorithms for locating the optimal feature sets, feature set prioritisation, multi objective search and feature interaction.

References

[1] G. Antoniol, M. Di Penta, and M. Harman. A robust search–based approach to project management in the presence of abandonment, rework, error and uncertainty. In 10^{th} *International Software Metrics Symposium (METRICS 2004)*, pages 172–183, Los Alamitos, California, USA, Sept. 2004. IEEE Computer Society Press.

[2] G. Antoniol, M. D. Penta, and M. Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. In 21^{st} *IEEE International Conference on Software Maintenance*, pages 240–249, Los Alamitos, California, USA, 2005. IEEE Computer Society Press.

[3] A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, Dec. 2001.

[4] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis. Software component ranking as feature subset selection. Seattle, Washington, USA, July 2006. To appear.

[5] A. W. Brown, editor. *Component-Based Software Engineering*. IEEE Press, 1997.

[6] F. Chicano and E. Alba. Management of software projects with gas. In 6^{th} *Metaheuristics International Conference (MIC2005)*, Vienna, Austria, Aug. 2005.

[7] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *IEE Proceedings — Software*, 150(3):161–175, 2003.

[8] M. Harman and B. F. Jones. Search based software engineering. *Information and Software Technology*, 43(14):833–839, Dec. 2001.

[9] G. T. Heineman and W. T. Councill, editors. *Component-Based Software Engineering*. Addison Wesley, 2001.

[10] C. Kirsopp, M. Shepperd, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1367–1374, San Francisco, CA 94104, USA, 9-13 July 2002. Morgan Kaufmann Publishers.

[11] M. M. Lehman. Software's future: Managing evolution. *IEEE Software*, 15(1):40–44, Jan. / Feb. 1998.

[12] Nashat Mansour, Rami Bahsoon and G. Baradhi. Empirical comparison of regression test selection algorithms. *Systems and Software*, 57(1):79–90, 2001.

$\|S_{ER}\|$	Budget	C_{GR}	C_{SA}	z_{ER}	z_{GR}	z_{SA}	$\|S_{GR}\|$	$\|S_{SA}\|$	$\|S_{GR} \bigcap S_{ER}\|$	$\|S_{SA} \bigcap S_{ER}\|$
1	100	100	100	1.00	9.00	13.50	3	7	0	0
2	150	150	150	2.00	6.00	15.50	2	7	0	0
3	450	450	440	5.00	9.00	26.00	3	13	1	1
4	530	530	520	8.00	18.00	27.50	6	14	2	1
5	600	600	590	11.00	23.50	28.83	9	16	3	3
6	700	700	700	12.00	25.50	30.00	10	15	3	3
7	1700	1700	1690	13.00	35.00	40.33	15	23	4	5
8	1740	1740	1740	14.50	34.00	40.58	14	24	3	7
9	1940	1940	1920	16.00	39.50	42.33	19	25	5	7
10	1960	1960	1960	16.50	39.00	42.58	18	26	6	8
11	3060	3040	3040	18.00	48.42	48.42	30	30	9	9
12	3070	3070	3040	21.00	37.50	48.42	16	30	7	10
13	3570	3570	3560	24.00	46.83	49.42	26	31	12	11
14	3580	3570	3560	25.00	46.83	49.42	26	31	13	12
15	3590	3570	3560	28.00	46.83	49.42	26	31	14	13
16	3600	3600	3600	30.00	47.00	49.50	26	31	15	14
17	3620	3620	3600	30.33	47.33	49.50	27	31	16	15
18	3820	3810	3640	30.67	46.50	49.75	25	32	15	16
19	4820	4810	4740	31.67	47.50	51.25	26	33	16	17
20	4940	4930	4740	32.67	49.00	51.25	27	33	16	18
21	5240	5220	4740	33.67	50.67	51.25	31	33	20	19
22	5290	5260	4740	34.17	50.92	51.25	32	33	21	20
23	5300	5300	4740	35.17	51.00	51.25	32	33	22	21
24	5330	5330	4740	36.67	50.00	51.25	28	33	19	22
25	5440	5430	5440	38.67	47.50	51.42	26	33	20	23
26	5670	5670	5660	40.67	49.83	51.92	29	33	23	25
27	5710	5700	5700	41.67	50.00	52.00	29	33	24	26
28	5890	5870	5740	43.67	49.83	52.25	29	34	26	27
29	5910	5900	5740	45.67	50.00	52.25	29	34	27	28
30	6060	6060	5740	47.67	51.58	52.25	32	34	29	29
31	6120	6100	5740	50.67	51.67	52.25	32	34	30	30
32	6220	6220	5740	51.67	51.67	52.25	32	34	32	31
33	6620	6620	6540	52.67	52.67	52.92	33	34	33	32
34	6700	6700	6700	53.00	53.00	53.00	34	34	34	34
35	6740	6740	6740	53.25	53.25	53.25	35	35	35	35

Figure 4: Results from the Simulated Annealing, Greedy and Expert Rankings and Selections

[13] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[14] G. Rothermel, S. Elbaum, A. G. Malishevsky, P. Kallakuri, and X. Qiu. On test suite composition and cost-effective regression testing. *ACM Trans. Softw. Eng. Methodol.*, 13(3):277–331, 2004.

[15] M. J. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(11):736–743, 1997.

[16] M. J. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In 18^{th} *IEEE International Conference on Software Engineering*, Los Alamitos, California, USA, Mar. 1996. IEEE Computer Society Press.

[17] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In P. G. Frankl, editor, *Proceedings of the ACM SIGSOFT 2002 International Symposium on Software Testing and Analysis (ISSTA-02)*, volume 27, 4 of *Software Engineer Notes*, pages 97–106, New York, July 22–24 2002. ACM Press.

[18] K. Steinhfel, A. Albrecht, and C. Wong. Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research*, 118(3):524 – 548, 1999.

[19] W. E. Wong, J. R. Horgan, S. London, and H. A. Bellcore. A study of effective regression testing in practice. In *ISSRE '97: Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE '97)*, page 264. IEEE Computer Society, 1997.