# Search–Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project

Giulio Antoniol* and Massimiliano Di Penta* and Mark Harman**
antoniol@ieee.org dipenta@unisannio.it Mark@dcs.kcl.ac.uk

* RCOST - Research Centre on Software Technology
University of Sannio, Department of Engineering
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy

**Department of Computer Science,
King's College London, Strand, London WC2R 2LS, UK

## Abstract

*This paper evaluates the use of three different search–based techniques, namely genetic algorithms, hill climbing and simulated annealing, and two problem representations, for planning resource allocation in large massive maintenance projects. In particular, the search–based approach aims to find an optimal or near optimal order in which to allocate work packages to programming teams, in order to minimize the project duration.*

*The approach is validated by an empirical study of a large, commercial Y2K massive maintenance project, which compares these techniques with each other and with a random search (to provide base line comparison data).*

*Results show that an ordering-based genome encoding (with tailored cross over operator) and the genetic algorithm appear to provide the most robust solution, though the hill climbing approach also performs well. The best search technique results reduce the project duration by as much as 50%.*

**Keywords:** Massive Remedial Maintenance, Search-Based Software Engineering

## 1 Introduction

In software maintenance, as in other large scale engineering activities, effective project planning is essential. Failure to plan and/or poor planning can cause delays and costs that, given timing and budget constraints, are often unacceptable, leading to business–critical failures. Traditional tools such as the Project Evaluation and Review Technique (PERT), the Critical Path Method (CPM), Gantt diagrams and Earned Value Analysis help to plan and track project milestones. While these tools and techniques are important, they cannot assist with the identification of optimal scheduling assignment in the presence of configurable resource allocation.

Furthermore, maintenance projects have their own characterisations and properties that require particular attention. Most large–scale maintenance interventions involve several teams of programmers and many individual project Work Packages (WPs). As such, the optimal allocation of teams of programmers (the primary resource cost drivers) to WPs is an important problem which cannot be overlooked.

This paper studies the problem of resource allocation in the context of a large–scale massive maintenance intervention. It investigates the use of search–based software engineering [23] techniques to address problems associated with optimal allocation of WPs. Specifically, the paper addresses problems associated with the determination of the optimal order in which to process WPs (and their consequent allocation to teams of programmers).

From the perspective of search–based software engineering, this is an example of a scheduling problem. Scheduling problems, of both general and specific characters, have received a great deal of attention from the search community, yet there has, to date, been little work on the application of search to the problem of software project planning.

In order to validate the techniques introduced in this paper, empirical results are presented from a large-scale case study of a massive maintenance project from a large European financial organisation. The case study concerns remedial work arising from the need to address the Y2K problem. The project involves 84 WPs, each of which involve the maintenance and reverse engineering of approximately 300 Cobol and JCL files.

Such 'massive maintenance' projects are becoming sur-

*Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*
*1063-6773/05 $20.00 © 2005* **IEEE**

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

prisingly prevalent. They involve maintenance and reverse engineering on a wide scale, across the organisation, platforms and systems for both code, design and data. Recent examples include Y2K remediation, Euro currency conversion, zip code and phone numbering changes [25]. Such maintenance activities present particularly acute problems for managers, since they have fixed hard deadlines and cut right across an entire software portfolio, touching almost every software asset possessed by the organisation. Fortunately, as this paper will demonstrate, some of the features of the planning process for these massive maintenance projects lend themselves to solution by search techniques.

The primary contributions of this paper are as follows:

- the paper shows how three different search algorithms, namely Genetic Algorithms (GAs), hill climbing and simulated annealing, can be applied to maintenance project process planning and studies two different representations of the problem. The results are encouraging. In comparison with the real-world data from the project, the search-based optimizations were capable of reducing the duration of the maintenance process by as much as 50%, simply by finding more optimal allocations;
- for each search technique, the paper compares the effectiveness of two different genome encodings; and
- the paper also presents results of a separate empirical study into the effect upon completion time of changes in staffing levels.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of the software maintenance process studied in the paper. Section 3 explains the three search algorithms and two problem representations that we implemented to study the application of search to massive maintenance planning, while Sections 4 and 5 present the empirical results of the paper. In particular, Section 4 investigates the optimal choice of search technique and representation, while Section 5 presents the results of the effect of different allocations of staff to project teams, comparing the results obtained by search with the actual project staffing and timeline. Section 6 presents related work and Section 7 concludes.

## 2 Context: The Massive Maintenance Project

The application we studied was a large scale financial system for which Y2K remediation was required. The maintenance task was decomposed into WPs, i.e., loosely coupled, elementary units (from one to nine for each application) subject to maintenance activities; each WP was managed by a WP leader and assigned to a maintenance team (the average team size was approximately four programmers). Overall, the entire system was composed of 84 WPs, each of which comprised, on average, of 300 COBOL and JCL files.

The project followed a phased maintenance process (similar to that defined by the IEEE maintenance standard [1]), encompassing five macro-phases:

1. *Inventory*: deals with the decomposition of the application portfolio into independent applications, and successive decomposition of each application into WPs;
2. *Assessment*: identifies, for each WP, candidate impacted items, using automatic tools;
3. *Technical Analysis (TA)*: deals with the analysis of impacted items and identifies a candidate solution among a set of pre-defined solution patterns;
4. *Enactment (Enact)*: an automatic tool was used to apply patches to the problems identified and analyzed in the previous phases. The solution was usually based on windowing [31];
5. *Unit Testing (UT)* was performed on each impacted WP; tools were used for automatic generation of test cases.

Because the intervention was performed almost semi-automatically and involved highly standardized activities, it was possible to make an assumption that it is valid to interchange between people and months. That is, given a maintenance team size, $s$ and the effort required $e$, the time $t$ necessary to perform the task is:

$$t \simeq \frac{e}{s} \tag{1}$$

Due to Brooks' law [7], this could be an overly optimistic assumption. However, as other authors have noted [2], given the small team sizes (fewer than eight people) and the standard (training–free) nature of the maintenance task, this approximation can be considered reasonable. The model can be generalized to situations in which Brooks' law does apply by the simple introduction of a non-linearity factor. A further simplification introduced with our study is the absence of dependencies between WPs (i.e., there is no constraint on possible orderings which can be selected).

The research questions this paper seeks to answer, by the application of search techniques to the planning of this maintenance task are as follows:

- For a fixed staffing level, what is the optimal order in which to present the WPs for action?
- How do the results vary with team size and distribution?



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

- What is the difference between GA, hill climbing and simulated annealing, both in terms of result quality and number of required fitness evaluations?
- Which is the best genome representation for the problem described in this paper?

# 3 The Different Approaches

This section explains how we formulated the problem as a search–based problem, using two different representations, GAs, hill climbing and simulated annealing. A random search was also implemented for each encoding, to provide base line (worst case) data.

The overall implementation, combining the search-based heuristics (GA, hill climbing, simulated annealing) and the two representations is shown in Figure 1. The two different schemas of encoding (and associated fitness functions) are the *pigeon hole* representation and the *ordering* representation. These are described in Section 3.1, while Section 3.2 describes the algorithms constructed for Hill Climbing, Simulated Annealing and GAs. For further details the reader can refer to books such as the one by Michalewicz and Fogel [34].

## 3.1 Genome Encodings

### 3.1.1 The pigeon hole representation

The pigeon hole representation describes the solution as an array of $N$ integers, where $N$ is the number of WPs. Each value of the array indicates the team the WP is assigned to. The representation schema is shown in Figure 1-a. The fitness function (which is minimized) is simply the value of the project's overall deadline. That is, for a single-step/multi-server maintenance process, the maximum completion time among the different servers.

### 3.1.2 The ordering representation

The ordering representation also represents the problem as an $N$-sized array, but the value of a representation element indicates the position of the WP in the incoming queue, for a single-queue/multi-server queuing system. The representation schema is shown in Figure 1-b.

The fitness function takes as input, the representation (i.e., the WP sequence) and computes the finishing deadline using the queuing simulator of Antoniol et al. [3].

## 3.2 The Different Search Techniques

### 3.2.1 The Hill Climbing Approach

In hill climbing, the search proceeds from a randomly chosen point in the search space by considering the neighbours of the point. Once a fitter neighbour is found this becomes the current point in the search space and the process is repeated. If there is no fitter neighbour, then the search terminates and a maxima has been found (by definition). The approach is called hill climbing, because when the fitness function is thought of as a landscape, with peaks representing points of higher fitness, the hill climbing algorithm selects a hill near to the randomly chosen start point and simply moves the current point to the top of this hill (climbing the hill). Clearly, the problem with the hill climbing approach is that the hill located by the algorithm may be a local maxima, and may be far poorer, in terms of fitness, than the global maxima in the search space. However, hill climbing is a simple technique which is easy to implement and has been shown to be a useful and robust technique for the software engineering applications of modularization [19, 35] and cost-estimation [28].

The hill climbing approach constructed for software project planning used in this paper works as follows:

1. It starts with a initial solution where each WP is randomly assigned to a team;
2. It randomly takes a WP and assigns it to a new, randomly selected, team.
3. The new configuration is accepted if and only if it leads to a finishing deadline smaller than the one obtained with the previous configuration.
4. The algorithm iterates through step 2, either for a given number of times, or until the fitness function remains unchanged for a given number of iterations.

### 3.2.2 The Simulated Annealing Approach

Simulated annealing [33], like hill climbing, is a method of local search. However, simulated annealing has a 'cooling mechanism' (referred to as the 'temperature') which initially allows moves to less fit solutions. The effect of 'cooling' on the simulation of annealing is that the probability of following an unfavorable move is reduced. This (initially) allows the search to move away from local optima in which the search might be trapped. As the simulation 'cools' the search becomes more and more like a simple hill climb.

When minimizing, the objective function is usually referred to as a cost function. When maximizing it is usually referred to as a fitness function.

The simulated annealing approach constructed for software project planning used in this paper works as follows: From the current assignment of WPs to teams, select a WP and assign it to a new team. This produces a move to a near neighbour in the search space. Unlike hill climbing, however, if the neighbour has a higher value for the objective



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

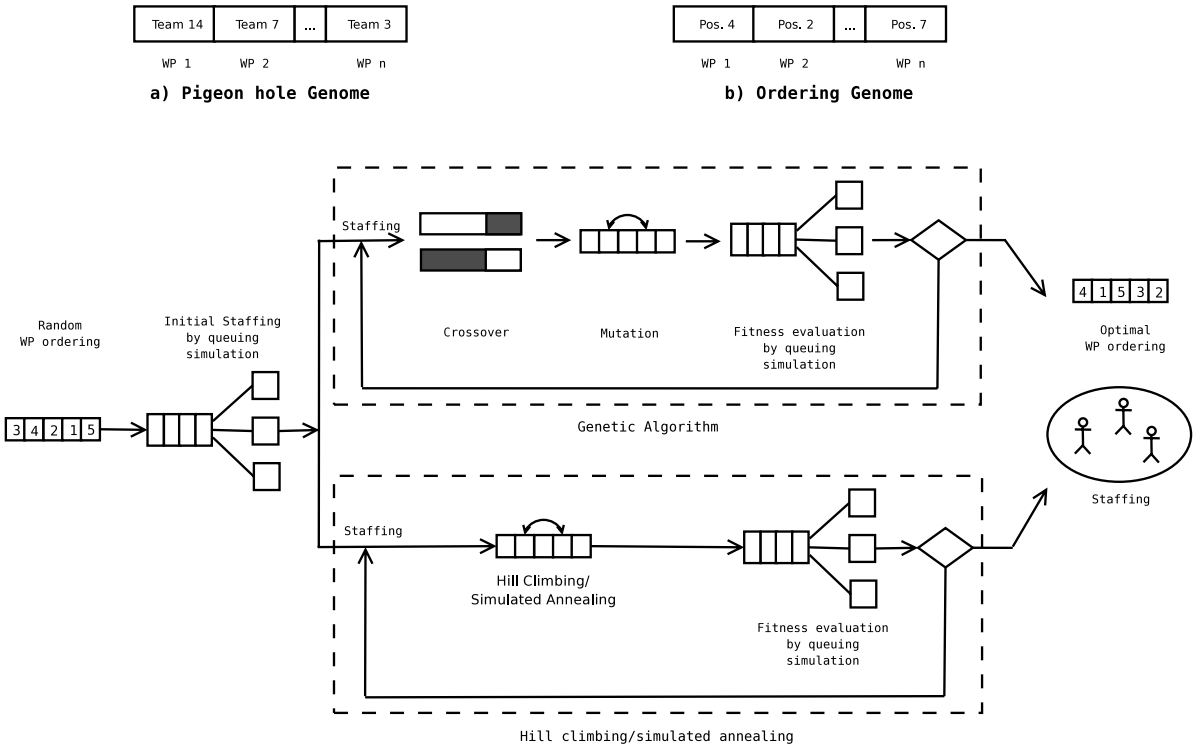


**Figure 1. a) The Pigeon Hole representation - b) The Ordering representation - c) The proposed approach**

function (the value of which is to be minimized), the new configuration *can* be accepted if:

$$p < m \tag{2}$$

where:

- $p$ is a random number in the range $[0 \ldots 1]$ and
- $m = e^{\Delta fitness/t}$

where $t$ (the fitness *temperature*) was chosen as

$$t = \frac{\alpha}{log(x + \beta)} \tag{3}$$

$\alpha$ and $\beta$ are constants of the same order of magnitude of $\Delta fitness$, and $x$ is the current number of iterations.

This ability to select possibly worse solutions, allows the simulated annealing approach to avoid local optima.

### 3.2.3 The Genetic Algorithm Approach

GAs originate with an idea, born over 30 years ago, of applying the biological principle of evolution to artificial systems. Roughly speaking, a GA may be defined as an iterative procedure that searches for the best solution of a given problem among a population, represented by a finite string of symbols, the *genome*. The search is made starting from an initial population of individuals, often randomly generated. At each evolutionary step, individuals are evaluated using a *fitness function*. High-fitness individuals will have the highest probability to reproduce.

The evolution (i.e., the generation of a new population) is made by means of two operators: the *crossover operator* and the *mutation operator*. The crossover operator takes two individuals (the *parents*) of the old generation and exchanges parts of their genomes, producing one or more new individuals (the *offspring*). The mutation operator has been introduced to prevent convergence to local optima; it randomly modifies an individual's genome (e.g., by flipping some of its bits, if the genome is represented by a bit string). Crossover and mutation are performed on each individual of the population with probability *pcross* and *pmut* respectively, where $pmut \ll pcross$. Further details on GAs can be found in the literature [15, 17].



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

GA are effective in finding optimal (or near optimal) solutions for problems where:

- the search space is large or complex;
- no mathematical analysis is available;
- traditional search methods did not work; and, as is the case for the problems considered in the present paper
- the solution of the problem is NP-hard [16].

A GA is not guaranteed to converge: the termination condition is often specified as a maximal number of generations, or as a given value of the fitness function. To implement a GA for the pigeon hole encoding, the mutation operator randomly selects a WP and randomly changes its team. The crossover operator is the standard *single point* crossover.

To implement a GA for the ordering encoding, the mutation operator randomly selects two WPs (i.e., two array items) and exchanges their position in the queue. The crossover operator is somewhat more complex. Two offspring ($o_1$ and $o_2$) are formed from two parents ($p_1$ and $p_2$), as follows:

1. A random position $k$, is selected in the genome.
2. The first $k$ elements of $p_1$ become the first $k$ elements of $o_1$.
3. The last $N$-$k$ elements of $o_1$ are the sequence of $N$-$k$ elements which remain when the $k$ elements selected from $p_1$ are removed from $p_2$.
4. $o_2$ is obtained similarly, composed of the first $N$-$k$ elements of $p_2$ and the remaining elements of $p_1$ (when the first $N$-$k$ elements of $p_2$ are removed).

For instance, if $k = 2$ and $p_1 \equiv \{4, 2, 3, 6\}$ and $p_2 \equiv \{4, 6, 3, 2\}$, then $o_1 \equiv \{4, 2, 6, 3\}$ and $o_2 \equiv \{4, 6, 2, 3\}$.

This approach to crossover has the advantage that it guarantees that each offspring contains precisely one position in the sequence per WP. It therefore avoids the need for repair, or some other mechanism which might be required to deal with duplication of sequence numbers in a more simple-minded crossover operator.

## 4 Empirical Study I: Comparing Search Techniques and Representations

First and foremost, we will analyze the difference i) between the two types of genome, i.e., the ordering genome and the pigeon hole genome, and ii) between all the different approaches described in Section 3. Results for all eight possible combinations are shown in Figure 2. For robustness, the average over ten runs for each encoding/search algorithm was used to produce the results reported in Figure 2. For the GA, the following parameters were chosen:

- Population size: 100 individuals;
- Type of GA: simple (non overlapping), however with elitism keeping the best individual alive over subsequent generations;
- Crossover probability: 0.6;
- Mutation probability: 0.1; and
- Roulette–wheel selection.

The comparison was performed considering a queuing network configuration with 20 teams (servers) composed of 1 person each one.

The figure clearly shows that the ordering genome encoding outperforms, in general, the pigeon hole genome encoding (the Mann–Whitney test showed that pigeon hole genome always performed significantly better, with a significance level of 95%). This supports the usefulness of the proposed approach that combines search–based heuristics with queuing simulation. The ordering genome is effective not only for speed of convergence, but also for the possibility it gives of modeling more complex maintenance tasks. A queuing simulator allows modeling multi–stage maintenance processes, even accounting for rework or abandonment after a given phase, as well as for priority queues and for dependencies between WPs.

The results also highlight the fact that applying search–based heuristics is a sensible approach because they significantly outperform a randomly-generated staff allocations. Only the GA–based pigeon hole algorithm reaches results comparable with the ordering genome, although requiring a high number of generations. The comparison of the different approaches, implemented using the ordering genome, highlights that the GA seems to exhibit a faster start (due to its intrinsic parallelism), although it is then overtaken by the hill climber. After 400/500 generations/iterations the results are exactly the same. Also the simulated annealing approach reaches, after about 100 generations, the same result of GA and hill climbing. It exhibits a slower start due to the likelihood (higher for the first generations) of accepting a worsening solution. The Mann–Whitney test showed no significant difference when comparing the three techniques.

## 5 Empirical Study II: Determining Staff Allocation to Teams

The remaining research questions from Section 2 are:



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

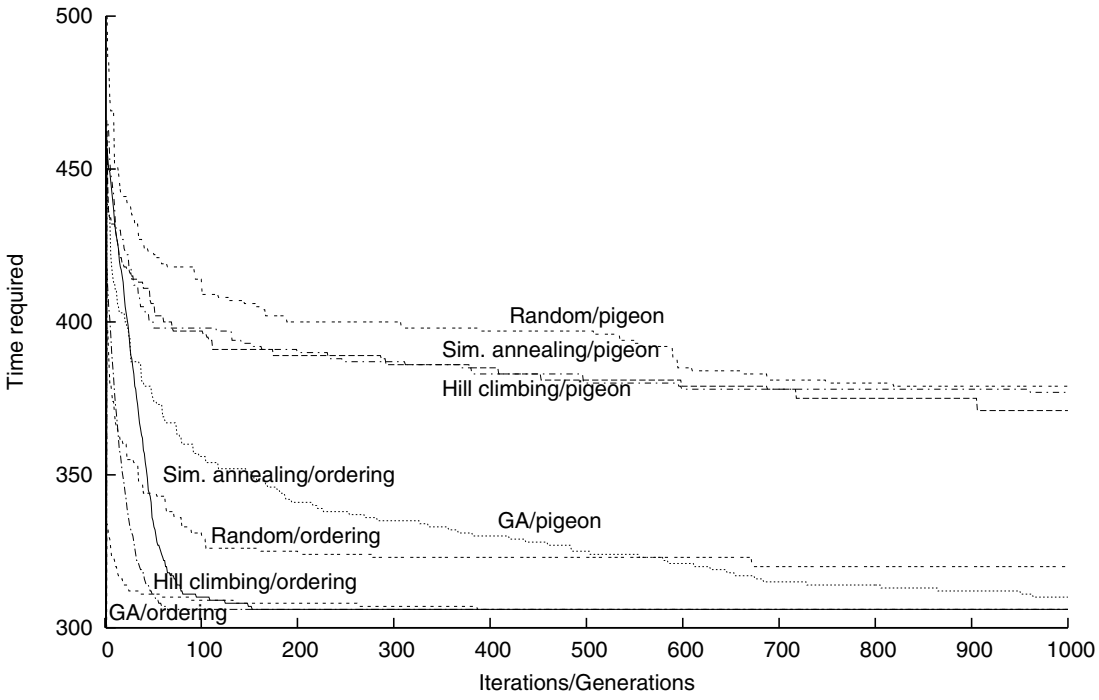


**Figure 2. Comparison of different genomes and different approaches**

1. How does the estimated finishing time vary with the number of people available? and

2. What happens if we consider groups of different size? That is, like Di Penta et al. [13], we may have some *fast lanes* in the queuing model. However, while in [13] the authors dedicated some servers to the shortest maintenance requests (similar to the fast lane checkout in a supermarket), here we can differently distribute the available people, having a given percentage of double–sized teams, i.e., *fast servers* to which the longest requests will be dispatched. As before, this assumes that Brooke's law is avoided because of the nature of the project (See Equation 1).

Results for different numbers of people available in total, and for different percentages of double–sized teams are shown in Figure 3. In particular, each line represents a different number (from 10 to 40) of people involved, the Y axis represents the result of the staffing (i.e., the estimated finishing time), while the X axis represents the percentage of double–sized teams. For instance, a percentage of 10% for a total of 10 people available means that 10% of the teams will be composed of two people instead of one only. In this case we will have a total of 9 teams, 8 composed of one person and one (about 10% of 9) composed of two people. As explained above, such a team can be used to handle (in less time) the longest requests; the fitness function will therefore tend to reward the assignment of a long task to a double–sized team, in that the resulting overall finishing time will be shorter. On the other hand, assigning short tasks to such a team will be almost useless, and could prevent the correct assignment of the longest tasks (by occupying the resource).

The figure shows that, for a small number of people available (between 10 and 15), it is not convenient to have any *fast servers*, since it implies a reduction of the total number of servers; for such a staffing level this is unacceptable. However, when the staffing increases, instead of excessively increasing the number of servers it turns out to be useful to have at least a small percentage of *fast servers*. These fast servers are able to prevent the longest task duration from determining the overall project finishing time. For example, let us suppose that the longest task requires 60 days, while all the remaining ones can be accomplished in 30 days. In that case, the total duration of the project cannot be smaller than 60 days. However, if instead, we double the size of the team working on that task, the total finishing time could be, for example, 40 days (because the longest task now completes in 30 days and that the reduction in the



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

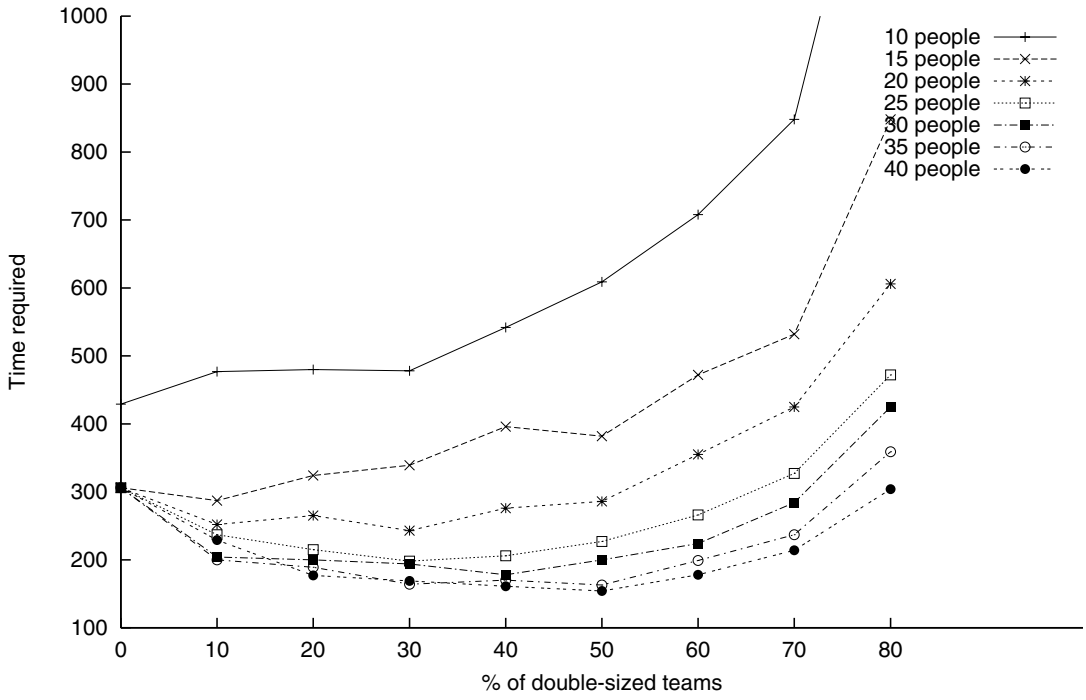


**Figure 3. Performance obtained with different numbers of people available overall for the project and different percentages of double–sized teams**

number of people available for the other tasks causes them to finish in 40 days).

A further increase of staffing tends to increase the optimal percentage of *fast servers*, moving the minimum point of the 'bathtub curve' to the right.

### 5.1 Actual project staffing

The actual staffing level of the project which was subject of our case study was 80 people. However, by analyzing the time sheets we noticed that these people did not work full–time on the project (i.e., the worked in parallel on other projects). The number of working teams and people allocated to teams varied during the project timeline, as also explained in previous work [3, 4]. In other words, people were moved from a team to another during the project. The number of days necessary to complete the project was of 155 working days.

Teams working on the different WPs varied in size from 2 to 27 (see Figure 4), with a median value of 6. Although the project manager did not adopt the "fast lane" model, at least they allocated some bigger teams to quickly maintain the bigger WPs. Finally, the number of teams working concurrently on the project (corresponding, in the queuing model, to the number of servers) varied from 1 to 12, with

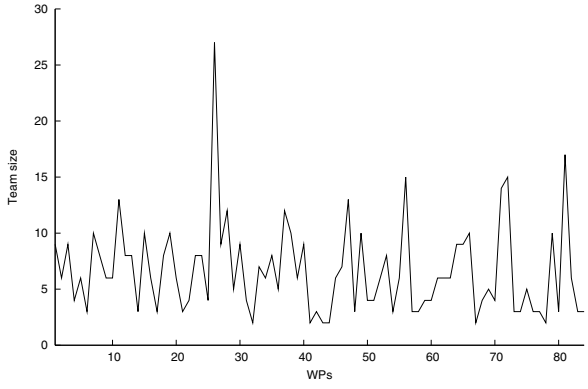


**Figure 4. Sizes of the teams working on the projects**



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

a median value of 6.

As shown in Figure 3, a staffing (working full time on the project) of 40 people, organized in 20 single-person team and 10 teams of two persons, would suffice to complete the project within the 155 days. This is in agreement with the result presented by Antoniol et al. [3], and confirms how the combination of queuing simulation and search heuristics can be used as a support for project management.

### 5.2 Threats to Validity

In a study like that reported in the present paper, there are three primary sources of treat to the validity of the results obtained. These concern the construct validity and internal and external validity of the results obtained. Construct validity concerns the methodology used to construct the experiment. Internal validity concerns possible courses of bias in the manner in which the results were obtained, while external validity concerns the possible bias of choice of experimental subjects (that is, to what extent can one generalised from these results).

Construct validity threats may be due to the simplifications made on the maintenance process modelled, as well as to the assumptions made in Section 2. However, the effect of these threats is limited because:

1. the applicability of the search techniques do not depend from the topology of the maintenance process (i.e., a more complex model simply requires that the objective function should be computed over a queuing network instead of over a single-queue system) and

2. for the case study adopted there were no dependencies at all.

Internal validity threats, in the case study, can be due to the randomness of the results obtained from simulation and GA. To avoid such a threat, different actions were taken:

- first and foremost, the number of generations and population size needed by GA were carefully calibrated. The chosen values were determined ensuring that further increases do not significantly affect the results;

- similarly, the number of iterations required by the whole approach was calibrated. Increases over 40 iterations do not produce improvements in the fitness function; and

- to avoid results being skewed by the randomness inherent in metaheuristic search techniques of the kind studied here, GAs were executed 20 times and it was verified that the fitness function values obtained by the final generation do not change among the iterations.

With regards to external validity, the approach and the results obtained can be extended *as is* to situations in which

1. Brooks' law is not valid and

2. there are no dependencies between WPs.

For all the other cases, there is the need for a more complex model. First and foremost, there may be the need to account for non-linearity factors in Equation (1), or to consider the constraints due to the WP dependencies.

## 6 Related Work

Search techniques have previously been applied to scheduling problems in general [11], with good results. The present authors introduced the idea of using search-techniques in software project planning [4]. However, this previous work only considered the application of GAs to the problem. The primary contribution of the present paper is to compare the use of GAs with a variety of other widely-used search techniques and encodings. This section explains the relationship between the work presented in the present paper and previous work on search–based scheduling, search based software engineering and software project planning.

An application of search–based techniques to project scheduling was done by Davis [11]. A survey of the application of GAs to solve scheduling problems has been presented by Hart et al. in [24]. The mathematical problem encountered is, as described by the author, the classical, NP-hard, bin packing or shop-bag problem. A survey of approximated approaches for the bin packing problem is presented in [10]. More recently Falkenauer published a book devoted to the GA and grouping problems [15]. The theme of the book and the proposed genome encoding are highly relevant to the problem addressed in this paper. Schema interpretation and bin packing genome encoding described in the book were a source of inspiration although our problem is slightly different. The order on which WPs are presented to the teams is relevant whereas bin packing doesn't impose a packing order to reach an optimum.

Search heuristics have been applied in the past to solve some related software project management problems. In particular, Kirsopp et al. reported a comparison of random search, hill climbing and forward sequential selection to select the optimal set of project attributes to use in a search–based approach to estimating project effort [27]. A comparison of approaches (both analytical and evolutionary) for prioritizing software requirements is proposed in [26], while Greer and Ruhe proposed a GA-based approach for planning software releases [18]. Release planning was also considered by Bagnall et al. [5], who applied a variety of techniques including greedy algorithms and simulated annealing to a set of synthetic data created to model features



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

for the next release and the relationships between them. The aim was to determine the subset of features which should optimally appear in the next release (the 'next release problem'), in order to maximize user satisfaction while minimizing cost. As such, their work can be viewed as an instantiation of a feature subset selection search problem, where as the set of problems considered in the present paper is characteristic of a scheduling search problem.

All of these approaches are examples of the wider class of problems embodied in the phrase 'search–based software engineering' [9, 23], which has become a fast growing subarea of activity with software engineering [8, 12, 22, 30], and into which category the work reported in the present paper also falls. Search based software engineering has proved successful in requirements engineering [5], project cost estimation [14, 29], testing [32] software maintenance and reverse engineering [20, 36] and program transformation [6, 21].

## 7 Conclusions

This paper has evaluated the use of search-based techniques for project planning in the context of a massive maintenance intervention. Three search based techniques were evaluated. Each was applied to two very different encoding strategies. Each encoding represents the way in which the WPs of the overall project are to be allocated to teams of programmers. The results from the case study show that the use of optimization can reduce project duration dramatically (by 50% over the results from the actual project).

The ordering encoding, which combines the search–based approach with a queuing simulation model, was found to outperform the other approaches. For the less optimal encoding, the GA performed significantly better than the other approaches. For the optimal encoding, though GA initially performs better, simulated annealing and hill climbing approaches soon catch up, so that the overall difference between the three approaches appears to be small, compared to the problem of establishing an effective encoding.

Finally, the paper reports the results of experiments that alter the size of the project teams. While, for a small overall staffing levels, double-sized teams do not improve performance, for larger staffing levels, the effect can be dramatic.

## 8 Acknowledgments

Mark Harman is supported, in part, by EPSRC grants GR/R43150 (FORTEST), GR/R98938 (TeTra), GR/S93684 (ASTRENET) and GR/T22872 (CONTRACTS) and by two development grants from DaimlerChrysler. The work reported here was also partly supported by EPSRC grant GR/M78083 (SEMINAL), which funded the SEMINAL network on Search-Based Software Engineering (SBSE). The authors are most grateful to the members of this network for many helpful discussions on SBSE.

## References

[1] *IEEE std 1219: Standard for Software maintenance*. 1998.

[2] T. Abdel-Hamid. The dynamics of software project staffi ng: a system dynamics based simulation approach. *IEEE Transactions on Software Engineering*, 15(2):109–119, 1989.

[3] G. Antoniol, A. Cimitile, G. A. Di Lucca, and M. Di Penta. Assessing staffi ng needs for a software maintenance project through queuing simulation. *IEEE Transactions on Software Engineering*, 30(1):43–58, Jan 2004.

[4] 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, Chicago, Illinois, USA, Sept. 2004. IEEE Computer Society Press, Los Alamitos, California, USA.

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

[6] A. Baresel, D. W. Binkley, M. Harman, and B. Korel. Evolutionary testing in the presence of loop–assigned flags: A testability transformation approach. In *International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 108–118, Omni Parker House Hotel, Boston, Massachusetts, July 2004. Appears in Software Engineering Notes, Volume 29, Number 4.

[7] F. Brooks. *The Mythical Man-Month 20th anniversary edition*. Addison-Wesley Publishing Company, Reading, MA, 1995.

[8] E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller. Search based software engineering track (LNCS 2724). In *GECCO 2003 Genetic and Evolutionary Computation*, Chicago, July 2003. Springer-Verlag.

[9] 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.

[10] E. J. Coffman, M. Garey, and D. Johnson. Approximation algorithms for bin-packing. *In Algorithm Design for Computer System Design*, 1984.

[11] L. Davis. Job-shop scheduling with genetic algorithms. In *International Conference on GAs*, pages 136–140. Lawrence Erlbaum, 1985.

[12] K. Deb, R. Poli, W. Banzhaf, H. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell. Search based software engineering track (LNCS 3103). In *GECCO 2004 Genetic and Evolutionary Computation*. Springer-Verlag, July 2004.



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.

[13] M. Di Penta, G. Casazza, G. Antoniol, and E. Merlo. Modeling web maintenance centers through queue models. In *European Conference on Software Maintenance and Reengineering*, pages 131–138, Lisbon, Portugal, March 2001. IEEE Society Press.

[14] J. J. Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021, 2000.

[15] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. Wiley-Inter Science, Wiley - NY, 1998.

[16] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[17] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub Co, Jan 1989.

[18] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, (to appear).

[19] M. Harman, R. Hierons, and M. Proctor. A new representation and crossover operator for search-based optimization of software modularization. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1351–1358, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[20] M. Harman, R. Hierons, and M. Proctor. A new representation and crossover operator for search-based optimization of software modularization. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1351–1358, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[21] M. Harman, L. Hu, R. M. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper. Testability transformation. *IEEE Transactions on Software Engineering*, 30(1):3–16, Jan. 2004.

[22] M. Harman and B. Jones. SEMINAL: Software engineering using metaheuristic innovative algorithms. In $23^{rd}$ *International Confernece on Software Engineering (ICSE 2001)*, pages 762–763, Toronto, Canada, May 2001. IEEE Computer Society Press, Los Alamitos, California, USA.

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

[24] E. Hart, D. Corne, and P. Ross. The state of the art in evolutionary scheduling. *Genetic Programming and Evolvable Machines*, 2004 (to appear).

[25] C. Jones. Mass-updates and software project management in is organizations- *http://www.artemis.it/artemis/artemis/lang_en/libreria/mass-updates.htm*, 1999. Accessed on Aug, 25 2003.

[26] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for priorizing software requirements. *Information and Software Technology*, 39:939–947, 1998.

[27] C. Kirsopp, M. Sheppard, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In *Genetic and Evolutionary Computation Conference*. Springer-Verlag, 2002.

[28] 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, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[29] C. Kirsopp, M. Shepperd, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1367–1374, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[30] W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska. Search based software engineering track. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, July 2002. Morgan Kaufmann Publishers.

[31] E. Lynd. Living with the 2-digit year, year 2000 maintenance using a procedural solution. In *Proceedings of IEEE International Conference on Software Maintenance*, pages 206–212, Bari, Italy, 1997.

[32] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verifi cation and Reliability*, 14(2):105–156, June 2004.

[33] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[34] Z. Michalewicz and D. Fogel. *How to solve it: Modern Heuristics (second edition)*. Springer, Berlin, 2004.

[35] B. S. Mitchell and S. Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1375–1382, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[36] B. S. Mitchell and S. Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1375–1382, New York, 9-13 July 2002. Morgan Kaufmann Publishers.



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 16, 2009 at 14:18 from IEEE Xplore. Restrictions apply.