A Multiple Hill Climbing Approach to Software Module Clustering

Kiarash Mahdavi
Mark Harman
Robert Mark Hierons
Department of Information Systems and Computing (DISC)
Brunel University
Uxbridge
Middlesex
UB8 3PH, UK
Kiarash.Mahdavi@brunel.ac.uk
Mark.Harman@brunel.ac.uk
Rob.Hierons@brunel.ac.uk

Keywords: Module clustering, Search based software engineering, Hill climbing.

Abstract

Automated software module clustering is important for maintenance of legacy systems written in a 'monolithic format' with inadequate module boundaries. Even where systems were originally designed with suitable module boundaries, structure tends to degrade as the system evolves, making re-modularization worthwhile. This paper focuses upon search-based approaches to the automated module clustering problem, where hitherto, the local search approach of hill climbing has been found to be most successful.

In the paper we show that results from a set of multiple hill climbs can be combined to locate good 'building blocks' for subsequent searches. Building blocks are formed by identifying the common features in a selection of best hill climbs. This process reduces the search space, while simultaneously 'hard wiring' parts of the solution.

The paper reports the results of an empirical study that show that the multiple hill climbing approach does indeed guide the search to higher peaks in subsequent executions. The paper also investigates the relationship between the improved results and the system size.

1 Introduction

It is generally believed that good modularization of software leads to systems that are easier to design, develop, test, maintain and evolve [1]. Modularisation of software (and the re-drawing of module boundaries) is also of crucial importance in software maintenance and evolution because it is well known that modular structure tends to decay as systems age, inhibiting efficient further maintenance and evolution.

The software module clustering problem consists of automatically finding a good quality clustering of software modules based on the relationships among the modules. These relationships typically take the form of dependencies between modules. The approach adapted in this paper for module clustering is to maximizes cohesion within each cluster and to minimize coupling between clusters. A clustering partitions the set of all modules in the system. The set of modules in each partition of the clustering is a cluster.

The problem of finding the best clustering for a given set of modules is an NP hard search problem [8], making it ideally suited to search-based software engineering techniques.

Previous work, most notably by the Drexel group [2, 6, 7, 8], has involved a hill climbing approach, which has been shown to produce reasonable quality clusterings. This work led to the development of the BUNCH tool for software module clustering.

Several studies have shown that for module clustering the results produced by hill climbing outperform standard global search techniques such as simulated annealing and genetic algorithms [2, 4, 9]. However, as is well known, hill climbing suffers from the problem of premature convergence to local optima and so it would be expected that some improvement could be made by considering more sophisticated search-based techniques.

In this paper we describe a multiple hill climbing approach. In this approach an initial set of hill climbs is performed and from these, a set of best hill climbs is identified according to some 'cut off' threshold. Using these

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

selected best hill climbs the common features of each solution are identified. These common features form building blocks for a subsequent hill climb.

A building block contains one or more modules fixed to be in a particular cluster, if and only if all the selected initial hill climbs agree that these modules were to be located within the same cluster. Since all the selected hill climbs agree on these choices, it is likely (though not certain) that good solutions will also contain these choices, hence the motivation for fixing them. In the nomenclature of search-based techniques these fixed module choices are 'building blocks' [10].

The implementation uses parallel computing techniques to simultaneously execute an initial set of 23 hill climbs. From these we experimented with various cut off points ranging from selecting the best 10% of hill climbs to the best 100% (effectively no cut off), in steps of 10%. This allowed us to evaluate the effect on the results when increasing and decreasing the selectivity.

The building blocks were fixed and a new set of 23 hill climbs were performed using the reduced search space. The principal research question is whether or not the identification of building blocks improved the subsequent search. We experimented with 19 programs, ranging from small systems with about 20 modules to large ones with over four hundred modules.

The results indicate that the subsequent search is narrowed to focus on better solutions, that novel and better clusterings are obtained and that the results tend to improve when the selection cut off is higher. These initial results are encouraging because they suggest that the multiple hill climbing technique is potentially a good way of identifying building blocks. This result may open the way for the successful application of more sophisticated global search techniques, such as genetic algorithms, to be applied in a hybrid approach which combines initial hill climbing and subsequent genetic search, seeded with the building blocks from the initial hill climbs. However the extension of this work to consider hybrid genetic and local search remains a problem for further study. The principal contributions of this paper are in the provision of empirical evidence that

- multiple hill climbs can be used to identify good building blocks;
- subsequent hill climbs find new peaks using the building blocks;
- selectivity appears to have a strong influence on the quality of results.

The study also raised the question as to whether the multiple hill climb technique would work better with larger systems than with smaller ones. This seemed intuitive, since larger systems are likely to be more complex and have more clustering choices, they would be likely to have more peaks. More peaks, would entail more chances to identify common features.

However, while we found that there was *some* correlation between system size and various measures of the improvement achieved with multiple hill climbing, none of these correlations was statistically significant.

The remainder of this paper is as follows. The multiple hill climb algorithm is explained in Section 2 followed by a description of how it was implemented in Section 3. The experiment is explained in Section 4 with the result and observations in Section 5. Sections 6 and 7 contain some conclusions drawn from the experimental results and possible future work respectively.

2 Multiple hill climb algorithm

The overall algorithm consists of an initial set of hill climbs, followed by the creation of building blocks which are used in the final set of hill climbs. The following explains these phases in more detail along with the fitness metrics used for the hill climb section of the algorithm.

2.1 Multiple hill climb algorithm's input

The algorithm uses *Module Dependency Graphs* (MDG) to as input for the hill climbers. Each MDG contains a list of from-to-weight information for the modules within the system to be clustered. This information is converted to a Vector of weighted connections between nodes. The weight is set to one if the weight value of a connection in the MDG is not specified.

2.2 Fitness metrics

The goal of module clustering is to arrive at a graph partition in which each cluster maximizes the number of internal edges and minimizes the number of external edges. In software engineering terms, this corresponds to maximal cohesion and minimal coupling [1].

In our approach, we use the 'Basic MQ' fitness function to capture this property as used by the Bunch team [8]. Basic MQ essentially captures this 'minimal coupling/maximal cohesion' metric. MQ is the sum of all Modularization Factors (MF). Each MF is based on the ratio of inner to outer edges within each module or group. An inner edge is a weighted connection from one node to another within the module. An outer edge is a weighted connection between a node within the module and a node outside of the module. This is demonstrated in the following.

$$i = 0 \Rightarrow MF = 0$$

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

$i > 0 \Rightarrow MF = \frac{i}{i+\frac{1}{2}j}$

Where i is the sum of inner edge weights
and j is the sum of outer edge weights.

The overall fitness MQ is calculated by:

$MQ = \sum_{i=1}^{n} MF_i$

Where i is a cluster
and n is the total number of clusters.

2.3 Initial set of hill climbs

Initially each module is assigned to a single building block. Since the MDG has N modules, there are can be up to N possible initial clusters. The initial hill climbs start by placing each building block at random in one of the N clusters. They then evaluate the fitness of the clustering resulting from this grouping by using MQ. Each hill climber attempts a move to a nearest neighbor clustering with a higher MQ at each stage of the algorithm. The nearest neighbours from each clustering are formed by moving a single building block from one module to another module. As soon as a fitter neighbour (neighbour with higher MQ) is found, the hill climber starts another search for a fitter neighbour from the newly found Clustering. The search ends when none of the nearest neighbours from a clustering can yield a better MQ value. This approach follows Mancoridis et al [6, 7].

2.4 Creating building blocks

Building blocks are the smallest units of change at each stage of the hill climb. The introduction of larger building blocks helps to reduce the search space for the hill climb algorithm with the aim of improving the search.

The clusterings from the first stage of the process are ranked by MQ and compared for similarity. The comparison identifies groups of nodes that are placed in the same cluster across a selected section of the initial clusterings. These selections are made from a proportion of the best hills climbed. The result is a set of building blocks, constructed from the initial set of hills found in the first phase.

2.5 Final set of hill climbs

Building blocks created from the initial set of hill climb results are used as nodes for the final set of hill climbs. The hill climb is identical to that used for the initial hill climb in Section 2.3.

3 Multiple hill climb implementation

This section briefly describes the parallel processing environment used and how the algorithm was implemented across this architecture.

3.1 Multi processor environment

The algorithm was implemented in Java on a `Scalable Linux Systems` (SCALI) at Brunel University called `GRIDS`. `GRIDS` contains 23 processing units (processing nodes) with a high speed processing node interconnection, and is accessed through a Linux Operating System interface.

3.2 Multi processor implementation

Each of the 23 processing nodes is set up as a server which carries out hill climb requests. A selected processing Node issues each server with a hill climb as necessary. This processing node also collects the clusterings resulted from the hill climbs carried out by the servers and identifies building blocks for further climbs. For simplicity we chose to keep together any modules which are in the same cluster across all the clusterings selected for similarity measurement. So, for example, if the 'cut off' point is 10%, then the modules are 'glued' together in the same building block if they are in the same cluster for *all* of the top 10% of hills found in the initial phase of hill climbs.

4 Experiment

The subject systems, information collected and the experimental procedure is described in this section.

4.1 Subjects

A variety of experimental subjects were used. Systems studied ranged in size from 20 modules to 413 modules. The MDGs representing the subjects were obtained by software analysis courtesy of the `Bunch` group at `Drexel` University. There are two types of MDG in this experiment. The first MDG type contains non-weighted edges while the second type contains weighted edges. Table 1 contains the names and short descriptions of the software used to create the MDGs and the number of nodes and edges within each MDG.

In graphs without weighted edges, each connection represents the existence of an unidirectional method or a variable reference between two modules. The MDGs containing specific values for weighted edges have the weights calculated according to the number of these unidirectional method and variable references between mod-

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

ules. Larger edge weights indicate more dependency between modules and an increase in the likely hood that they should be placed in the same cluster.

4.2 Procedure

Five experimental runs were carried out on each MDG. Each experiment, as described in section 2, consist of two stages. In the initial stage 23 initial hill climbs were carried out, one on each of the 23 processing units. The resultant clusterings were used to create the building blocks for the final stage of the Process. Building blocks were created based on the best 10% to 100% MQ values for the initial clusterings (in steps of 10%). Second stage is a final ten sets of hill climbs (for each top percentage clusterings used for building blocks) on the 23 processing units.

The first and second stage resultant clusterings along with the MQ achieved from each processing unit were collected. The MQ values achieved by the first and second stage were then compared and analyzed for their level of significant difference as well as other trends and correlations.

5 Results and observations

This section contains a summery of the results from the experiments and points to some of the trends and characteristics observed within these results.

5.1 Results

Figure 1 displays the best results and Figure 2 displays the worst results obtained by using the MDG's that do not have weighted edges. Figure 3 contains the best results and Figure 4 displays the worst results from MDGs with weighted edges.

These figures are represented as boxplots. The details on the axis of the boxplots is too small to read. However, the collection of distribution illustrated by boxplots gives an overall visual impression for the effects of the approach on the results. The right most boxplot shows the MQ values achieved by the initial hill climb. The other box plots from right to left show the MQ values achieved by using 100% to 10% of the initial climb results to create building blocks.

The boxplots have the following structure: The solid black, horizontal line represents the Median ($50^{th} percentile$). The top area within the box represents the upper quartile ($75^{th} percentile$) and the bottom area the lower quartile ($25^{th} percentile$). Circles represent outlier values, which are more than 1.5 box lengths above or below the box. Stars show Extreme values which are more than three box lengths above or below the box. Finally the horizontal thin lines above and below represent largest and smallest MQ values that are not outliers or extreme values.

In addition Wilcoxen signed ranked tests were used to check for significant differences between initial hill climbs and following hill climbs results (see Tables 2 and 3). It was also possible to use T-test to measure significant difference. In this case however due to the presence of outliers and the lack of evidence for normal distribution in some of the results the Wilcoxen test was used, since this test assumes neither normality or homogeneity of variance.

Table 4 contains the MQ increase from the best fitness achieved in the initial stage compared to the best fitness achieved at each final stage for weighted and non-weighted MDGs. Table 5 contains the same information as Table 4, represented as increased percentages to help achieve a fairer comparison of results. The range of values in Table 4 is better demonstrated by Figures 5 and 6 for MDGs with no weighted edges and 9 and 10 for MDGs with weighted edges, displayed against number of edges and nodes respectively. Similarly Figures 7, 8, 11 and 12 represent the range of percentage increase in MQ from Table 5.

5.2 Observations

The Wilcoxen signed ranked test provides some evidence towards the premise that the improvement in MQ values is less likely to be a random occurrence due to the nature of the hill climb algorithm. In general lower values demonstrate a higher level of certainty of a significant difference. For example 0.01% is statistically highly significant. Significant improvement in all hill climbs using building blocks at 10% and 20% is apparent (Tables 2, 3, 4 and 5). This improvement is observed for MDGs with and without weighted edges and for all size MDGs.

Larger size MDGs show more substantial improvement when the best initial fitness is compared with the best final fitness values. This improvement is even more apparent in very large MDGs such as the one for `swing` (best and worst performance for `swing` in Figures 1 and 2) and `nmh` (best and worst performance for `nmh` in Figures 3 and 4). On the other hand, for small MDG's of 20 to 30 nodes we observed less improvement in the final runs. One possible explanation is the less complex solution landscape of smaller systems. The initial hill climbs are more likely to find good peaks or sometimes the best peaks in the landscape resulting in less likelihood of improvement in following runs of the hill climb. However the reduction in variance helps the search to achieve consistently better results (for example best and worst performance for `ispell` shown in Figures 3 and 4). In ad-

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

	Name	nodes	edges	Description
Not Weighted	mtunis	20	57	An operating system for educational purposes written in the Turing language.
	ispell	24	103	Software for spelling and typographical error correction in files.
	rcs	29	163	Revision Control System used to manages multiple revisions of files.
	bison	37	179	General-purpose parser generator for converting grammar descriptions into C programs.
	grappa	86	295	Genome Rearrangements Analyzer under Parsimony and other Phylogenetic Algorithms.
	bunch	116	365	Software Clustering tool(Essential java classes only).
	incl	174	360	Graph drawing tool.
	bunchall	324	1344	Software Clustering tool(bunch + all related Java classes).
	swing	413	1513	Integration software for Lotus notes and Microsoft office.
Weighted	exim	23	1255	Message transfer agent for use on Unix systems connected to the Internet.
	bitchx	23	1653	Open source IRC client.
	lynx	23	1745	Web browser for users on UNIX and VMS platforms.
	icecast	60	650	Streaming media server based on the MP3 audio codec.
	gnupg	88	601	Complete implementation of the OpenPGP Internet standard.
	inn	90	624	Unix news group software.
	xntp	111	729	Time synchronization tool.
	mod_ssl	135	1095	Apache SSL/TLS Interface.
	ncurses	138	682	Software for Display and update of text on text-only terminals.
	nmh	198	3262	Mail client software.

Table 1. MDG's with and without weighted edges

	Name	nodes	edges	Significant difference with initial at %									
				100	90	80	70	60	50	40	30	20	10
Best	mtunis	20	57	.412	.420	.626	.821	.961	**.006**	**.005**	**.000**	**.000**	**.000**
	ispell	24	103	.033	.023	.168	.013	**.003**	**.010**	**.009**	**.000**	**.000**	**.000**
	rcs	29	163	.033	.023	.168	.013	**.003**	**.010**	**.009**	**.000**	**.000**	**.000**
	bison	37	179	.465	.346	.153	.627	.715	.107	.248	**.006**	**.001**	**.000**
	grappa	86	295	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	bunch	116	365	.951	.784	.394	.563	.976	.394	**.000**	**.000**	**.000**	**.000**
	incl	174	360	.378	.484	.903	.394	.605	**.000**	**.000**	**.000**	**.000**	**.000**
	bunchall	324	1344	.007	.018	**.007**	**.001**	**.000**	**.002**	**.000**	**.000**	**.000**	**.000**
	swing	413	1513	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
Worst	mtunis	20	57	.370	.783	.140	.144	.144	.236	.079	.121	**.000**	**.000**
	ispell	24	103	.068	.201	.091	.023	.362	.017	.224	**.010**	**.002**	**.000**
	rcs	29	163	.171	**.010**	.013	.083	**.004**	**.003**	.073	.019	**.009**	**.000**
	bison	37	179	.693	.927	.879	.394	.447	.808	.927	.018	**.000**	**.000**
	grappa	86	295	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	bunch	116	365	.670	**.007**	.563	.201	.260	.584	.465	1.000	**.000**	**.000**
	incl	174	360	.715	.181	.855	.114	.506	.301	.484	.784	**.000**	**.000**
	bunchall	324	1344	**.003**	**.003**	**.007**	**.001**	**.001**	**.001**	**.001**	**.001**	**.004**	**.000**
	swing	413	1513	**.000**	**.000**	**.000**	**.000**	**.003**	**.000**	**.000**	**.000**	**.000**	**.000**

Table 2. Wilcoxon signed ranked test results of significant difference against initial hill climb results for MDGs with no weighted edges

	Name	nodes	edges	% of initial clustering used for building blocks									
				100	90	80	70	60	50	40	30	20	10
Best	exim	23	1255	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	bitchx	23	1653	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	lynx	23	1745	**.001**	.011	**.001**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	icecast	60	650	**.000**	**.000**	**.000**	**.001**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	gnupg	88	601	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	inn	90	624	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	xntp	111	729	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	mod_ssl	135	1095	.287	.171	.024	**.007**	.412	.260	**.000**	**.000**	**.000**	**.000**
	ncurses	138	682	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	nmh	198	3262	.879	.808	.201	.023	.083	**.000**	**.000**	**.000**	**.000**	**.000**
Worst	exim	23	1255	**.001**	**.000**	**.002**	**.002**	**.001**	**.004**	**.004**	**.005**	**.000**	**.000**
	bitchx	23	1653	**.000**	**.000**	**.001**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	lynx	23	1745	.026	**.007**	**.002**	**.002**	**.004**	**.002**	**.001**	**.000**	**.000**	**.000**
	icecast	60	650	**.001**	**.000**	**.000**	**.002**	**.000**	**.001**	**.000**	**.000**	**.000**	**.000**
	gnupg	88	601	**.001**	**.000**	**.003**	**.001**	**.000**	**.002**	**.000**	**.000**	**.000**	**.000**
	inn	90	624	**.000**	**.001**	**.002**	**.001**	**.001**	**.001**	**.001**	**.001**	**.001**	**.000**
	xntp	111	729	**.000**	**.000**	**.000**	**.002**	**.001**	**.002**	**.000**	**.000**	**.000**	**.000**
	mod_ssl	135	1095	.078	.083	.024	**.002**	.039	.012	.013	.029	.033	**.000**
	ncurses	138	682	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**	**.000**
	nmh	198	3262	.761	.976	.484	.465	.362	.670	**.005**	**.003**	**.000**	**.000**

Table 3. Wilcoxon signed ranked test results of significant difference against initial hill climb results for MDGs with weighted edges

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

	Name	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Not Weighted	mtunis	0.0279	0.0279	0.0279	0.0279	0.0279	0.0279	0.0279	0.0279	0.0279	0.0279
	ispell	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0	0.0006	0.0006
	rcs	0.0023	0.0054	0.0054	0.0054	0.0054	0.0054	0.0054	0.0054	0.0013	0.0033
	bison	0.0256	0.0379	0.0236	0.0175	0.0379	0.0070	0.0072	0.0062	0.0010	0.0214
	grappa	0.0453	0.1924	0.1729	0.1924	0.1924	0.1842	0.1924	0.1842	0.1842	0.1729
	bunch	0.1803	0.2940	0.0688	0.2422	0.0627	0.3197	0.0405	0.0578	0.0371	0.2565
	incl	0.1156	0.1468	0.1218	0.1186	0.1223	0.0629	0.1084	0.0568	0.1049	0.1218
	bunchall	0.1035	0.0885	0.0781	0.1033	0.0930	0.0695	0.0693	0.1312	0.0766	0.0570
	swing	1.1051	0.5117	0.5381	0.4811	0.5688	0.5787	0.7095	0.3641	0.7104	0.4653
Weighted	exim	0.0825	0.1235	0.1056	0.1189	0.1124	0.1148	0.1079	0.1041	0.0725	0.1099
	bitchx	0.0668	0.0668	0.0548	0.0550	0.0538	0.0615	0.0562	0.0489	0.0470	0.0500
	lynx	0.0241	0.0315	0.0316	0.0282	0.0466	0.0409	0.0312	0.0219	0.0363	0.0207
	icecast	0.0092	0.0177	0.0177	0.0176	0.0176	0.0177	0.0155	0.0176	0.0176	0.0155
	gnupg	0.0611	0.0744	0.0672	0.0824	0.0733	0.0596	0.0744	0.0676	0.0686	0.0733
	inn	0.3066	0.3049	0.2754	0.5772	0.7500	0.3049	0.7397	0.4193	0.3137	0.5544
	xntp	0.0630	0.0630	0.0564	0.0523	0.0617	0.0557	0.0612	0.0600	0.0483	0.0575
	mod_ssl	0.3140	0.3300	0.3179	0.3256	0.1211	0.0713	0.1558	0.2076	0.1361	0.1910
	ncurses	0.2068	0.2371	0.2270	0.2292	0.2202	0.2112	0.2218	0.2324	0.2444	0.2271
	nmh	0.0997	0.1838	0.1216	0.1270	0.0997	0.1018	0.1132	0.1273	0.1249	0.0910

Table 4. Increase in fitness from the best final stage's MQ value compared to best initial stage's MQ value.

	Name	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Not Weighted	mtunis	0.0122	0.0122	0.0122	0.0122	0.0122	0.0122	0.0122	0.0122	0.0122	0.0122
	ispell	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0	0.0002	0.0002
	rcs	0.0010	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0006	0.0014
	bison	0.0095	0.0142	0.0088	0.0065	0.0142	0.0026	0.0027	0.0023	0.0003	0.0080
	grappa	0.0035	0.0153	0.0138	0.0153	0.0153	0.0147	0.0153	0.0147	0.0147	0.0138
	bunch	0.0138	0.0228	0.0053	0.0188	0.0048	0.0248	0.0030	0.0045	0.0028	0.0199
	incl	0.0088	0.0111	0.0092	0.0090	0.0093	0.0047	0.0082	0.0043	0.0079	0.0092
	bunchall	0.0061	0.0052	0.0046	0.0061	0.0055	0.0041	0.0041	0.0078	0.0045	0.0033
	swing	0.0252	0.0117	0.0122	0.0109	0.0130	0.0132	0.0162	0.0083	0.0162	0.0106
Weighted	exim	0.0127	0.0190	0.0162	0.0183	0.0172	0.0176	0.0166	0.0160	0.0111	0.0169
	bitchx	0.0154	0.0154	0.0127	0.0127	0.0124	0.0142	0.0130	0.0113	0.0108	0.0115
	lynx	0.0048	0.0063	0.0064	0.0057	0.0094	0.0082	0.0063	0.0044	0.0073	0.0041
	icecast	0.0033	0.0065	0.0065	0.0064	0.0064	0.0065	0.0056	0.0064	0.0064	0.0056
	gnupg	0.0086	0.0105	0.0095	0.0116	0.0103	0.0084	0.0105	0.0095	0.0096	0.0103
	inn	0.0442	0.0424	0.0383	0.0835	0.1086	0.0424	0.1071	0.0607	0.0436	0.0802
	xntp	0.0076	0.0076	0.0068	0.0063	0.0074	0.0067	0.0073	0.0072	0.0058	0.0069
	mod_ssl	0.0324	0.0341	0.0329	0.0336	0.0125	0.0073	0.0162	0.0216	0.0141	0.0198
	ncurses	0.0178	0.0204	0.0196	0.0197	0.0190	0.0181	0.0191	0.0200	0.0211	0.0196
	nmh	0.0107	0.0199	0.0132	0.0138	0.0107	0.0110	0.0122	0.0138	0.0135	0.0098

Table 5. Percentage increase in the best final stage MQ fitness compared to best initial stage fitness value.

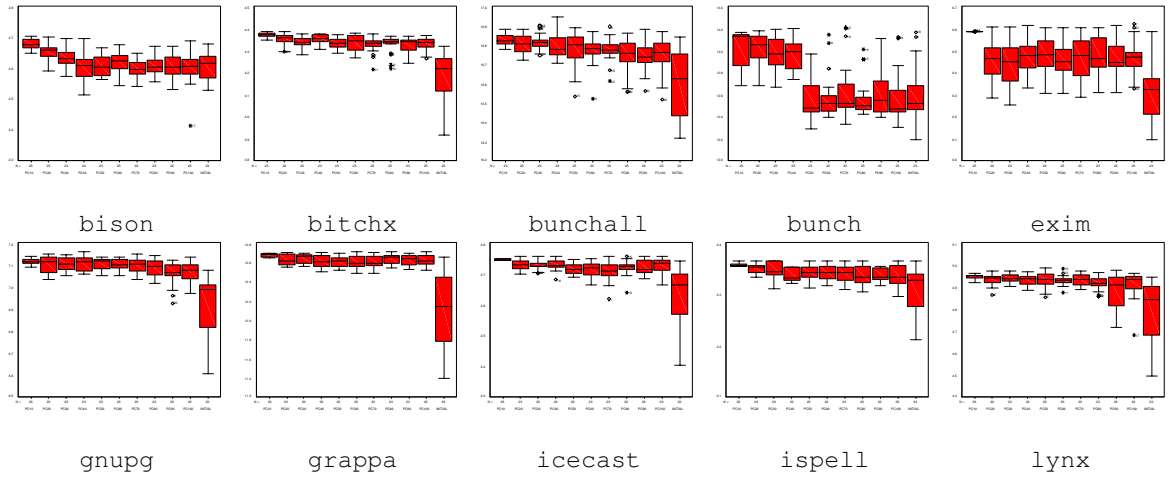

Figure 1. Best results obtained by using MDGs without weighted edges.

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

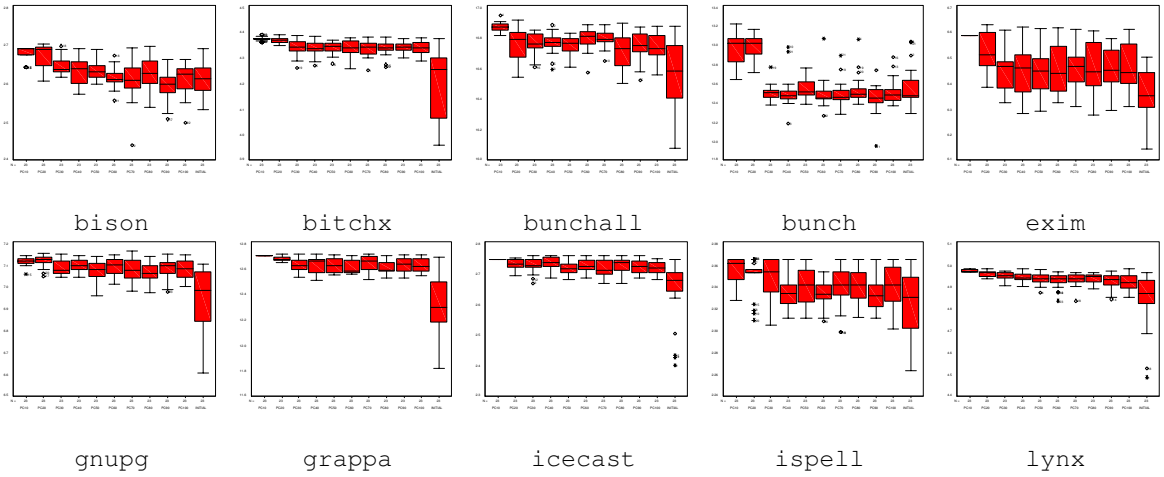

Figure 2. Worst results obtained by using MDGs without weighted edges.

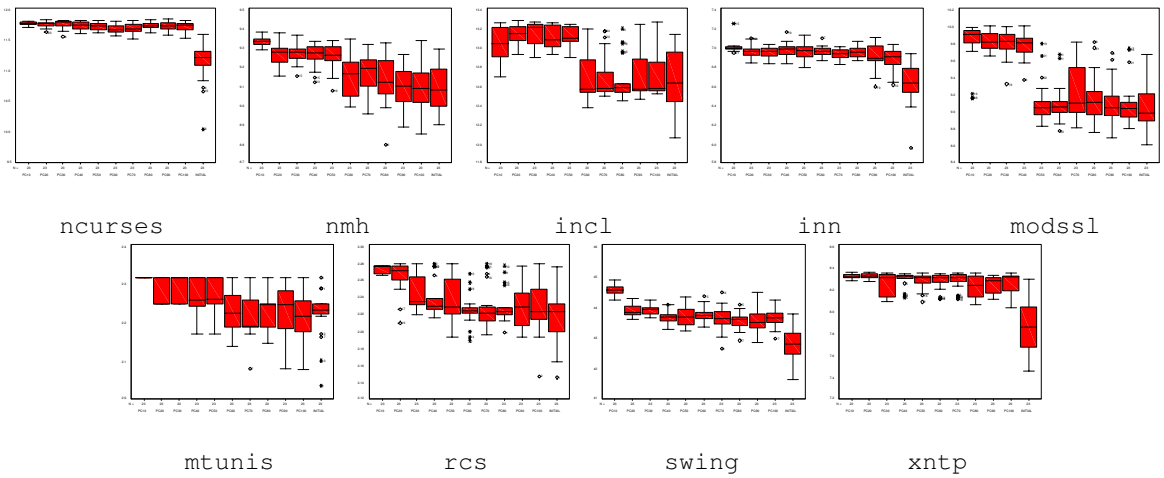

Figure 3. Best results obtained by using MDGs with weighted edges.

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

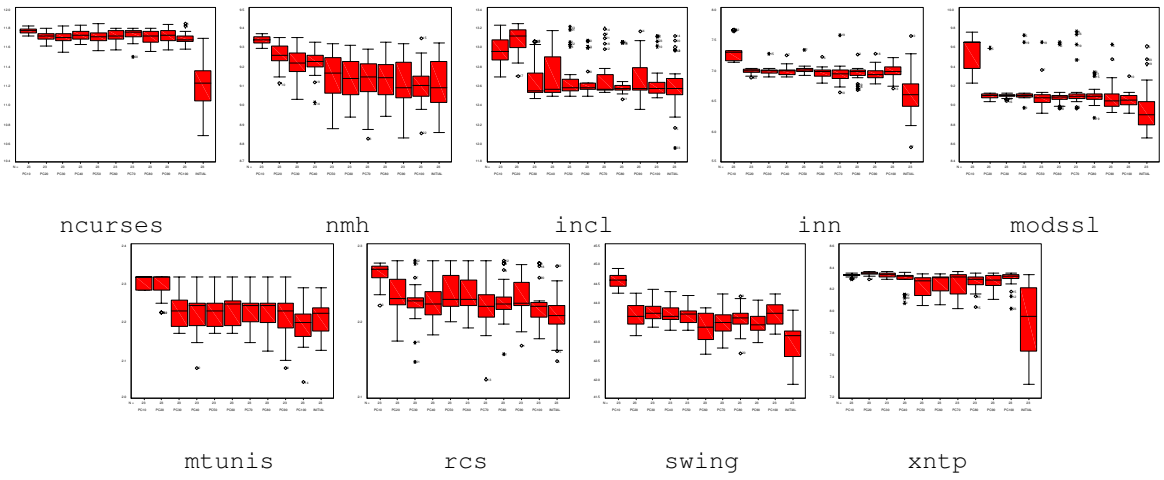

Figure 4. Worst results obtained by using MDGs with weighted edges.

dition, weak results from the initial hill climb can also achieve consistently better values (for example best and worst performance for `mtunis` shown in Figures 1 and 2).

One reason for observing more substantial improvement in larger MDGs may be attributed to the nature of the MQ fitness measure. Unfortunately the MQ fitness measure is not normalized, for example a double increase of MQ does not signify a doubling of modularization quality. At best, we can only claim that MQ is an ordinal metric [11]. To overcome this, the percentage MQ improvement of the final runs over the initial runs is also measured (see Table 5). Using these values, tests were carried out to determine any improvement correlating with the MDG complexity. The number of nodes and the number of connections in each MDG were tested for correlation against largest percentage improvement of each of the final runs against the initial run. These statistical tests show no significant correlation between size and improvement in fitness irrelevant of weighted or non-weighted MDGs.

Improvements are always achieved for selection cut off values of 10% and 20%, in most cases there are improvements across all final hill climbs. However there are exceptions. A dramatic example of this is in `bunch` (Figures 1 and 2), where results only show an increase for the cases where 10% and 20% of the initial climbs are used for building blocks.

5.3 Experimental concerns

Due to inherent randomness in any hill climbing search technique, it is hard to identify any trends by looking at individual hill climbs. For this reason multiple runs of the algorithm were used. Furthermore this technique was used on MDGs with weighted and without weighted edges of different sizes to improve the strength of the results for more general cases.

Employing the Wilcoxen signed ranked test helped to show that the improvements are significant enough to be an unlikely chance occurrence. The reduction in variance caused by the selection mechanism may mislead the Wilcoxen ranked test to find significant difference between the initial and final runs. Therefore actual improvement in the fitness over the initial runs were measured to determine whether the search is capable of discovering better peaks in the landscape.

6 Conclusions

The multiple hill climb technique proposed here has produced improved results across all MDGs, weighted and non-weighted. There is some evidence that the technique works better for larger MDGs but this could be due to the ordinal nature of the MQ metric used to assess modularisation quality.

This difficulty aside, larger MDGs tend to achieve relatively earlier benefits across the final hill climb runs from this technique. For example MDGs with small number of nodes and edges tend to show little or no improvement until building blocks used for the final hill climb are selected at 10% and 20%. On the other hand MDGs with a large number of nodes and edges tend to show significant improvement on the initial search across most or all of the final runs (Tables 2 and 3).

The increase in fitness, regardless of number of nodes or edges, tends to be more apparent as the building blocks are created from a smaller selection of individuals. This

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

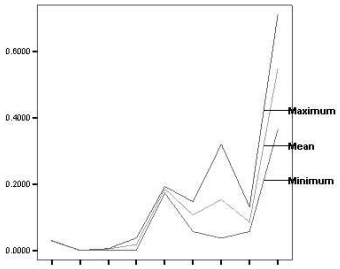

Figure 5. MQ increase against number of edges for MDGs with no weighted edges

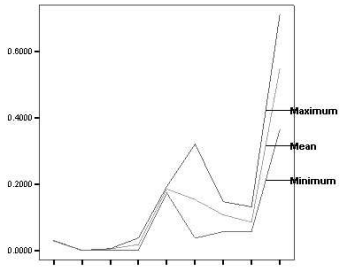

Figure 6. MQ increase against number Of nodes for MDGs with no weighted edges

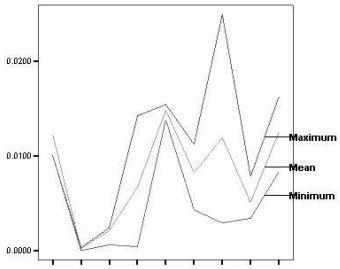

Figure 7. Percentage MQ increase against number Of edges for MDGs with no weighted edges

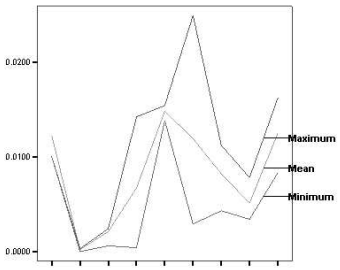

Figure 8. Percentage MQ increase against number Of nodes for MDGs with no weighted edges

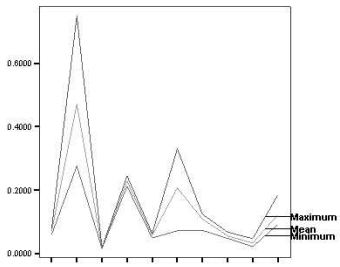

Figure 9. MQ increase against number Of edges for MDGs with weighted edges

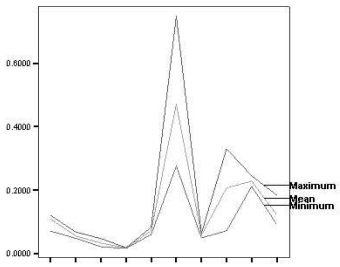

Figure 10. MQ increase against number Of nodes for MDGs with weighted edges

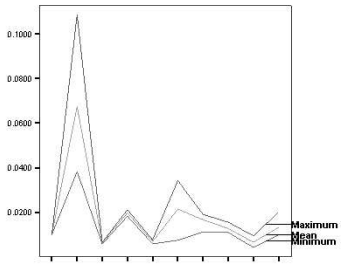

Figure 11. Percentage MQ increase against number Of edges for MDGs with weighted edges

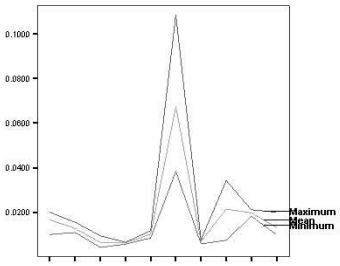

Figure 12. Percentage MQ increase against number Of nodes for MDGs with weighted edges

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE

may signify some degree of importance for the selection process. Perhaps the less fit solutions in the initial population are more likely to represent the same peak in the solution space and removing them by a more elite selection process may reduce the noise or bias this may introduce and increase the likelihood of a more concentrated search.

7 Future work

The selection techniques used in building block creation may be extended. This may well be achieved by ensuring that all hill climbs in the initial stage are unique. For very small MDGs this may cover all peaks in the landscape, also some sections of the landscape may be harder to search in the initial stage. Another possible method to ensure improved selection would be to include an attribute which determines the importance of each initial result in construction of the building blocks. This attribute could be related to frequency or distribution of the initial solutions.

In addition, other techniques to measure complexity of the MDGs and use of a normalised fitness measure could help in the recognition of any relationship between MDG complexity and the improvement achieved by using this technique.

Finally, once an improved selection technique is identified, multiple iterations of building block creation/ hill climbs can be used to focus the search further. Alternatively this technique could be used to improve genetic algorithms (GAs). GAs have already been used for clustering but with generally worse results than pure hill climbing [2, 4, 8]. The less than ideal results might be a consequence of the crossover operator in GAs, which is deemed to be more effective when the structure of the chromosomes aids the transmission of useful information between generations [5]. The use of the building blocks created using this technique to seed a GA could help to preserve this information and improve the GA's performance without resorting to complicated evolutionary repair operators [3].

8 Acknowledgements

We would like to thank Spiros Mancoridis and Brian Mitchell at Drexel university for their help with software clustering and Simon Taylor at Brunel university for allowing the work to be conducted on the GRIDS parallel processing system. We would also like to thank the members of the EPSRC Software Engineering Using Metaheuristic Algorithms Network and the Brunel Intelligent Data Analysis group for many helpful discussions about clustering.

This work is supported, in part, by EPSRC Grants GR/R98938, GR/M58719, GR/M78083 and GR/R43150 and by the Brunel Research Initiative and Enterprise Fund.

References

[1] CONSTANTINE, L. L., AND YOURDON, E. *Structured Design*. Prentice Hall, 1979.

[2] DOVAL, D., MANCORIDIS, S., AND MITCHELL, B. S. Automatic clustering of software systems using a genetic algorithm. In *International Conference on Software Tools and Engineering Practice (STEP'99)* (Pittsburgh, PA, 30 August - 2 September 1999).

[3] FALKENAUER, E. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation 2*, 2 (1994), 123–144.

[4] HARMAN, M., HIERONS, R., AND PROCTOR, M. A new representation and crossover operator for search-based optimization of software modularization. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference* (New York, 9-13 July 2002), Morgan Kaufmann Publishers, pp. 1351–1358.

[5] JONES, T. Crossover, macromutation, and population-based search. In *Proceedings of the 6th International Conference on Genetic Algorithms* (San Francisco, July 15–19 1995), L. J. Eshelman, Ed., Morgan kaufmann Publishers, pp. 73–80.

[6] MANCORIDIS, S., MITCHELL, B. S., CHEN, Y.-F., AND GANSNER, E. R. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings; IEEE International Conference on Software Maintenance* (1999), IEEE Computer Society Press, pp. 50–59.

[7] MANCORIDIS, S., MITCHELL, B. S., RORRES, C., CHEN, Y.-F., AND GANSNER, E. R. Using automatic clustering to produce high-level system organizations of source code. In *International Workshop on Program Comprehension (IWPC'98)* (Ischia, Italy, 1998), IEEE Computer Society Press, Los Alamitos, California, USA, pp. 45–53.

[8] MITCHELL, B. S. *A Heuristic Search Approach to Solving the Software Clustering Problem*. PhD Thesis, Drexel University, Philadelphia, PA, Jan. 2002.

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

[10] MITCHELL, M. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[11] SHEPPERD, M. J. *Foundations of software measurement*. Prentice Hall, 1995.

Proceedings of the International Conference on Software Maintenance (ICSM'03)
1063-6773/03 $17.00 © 2003 IEEE