Allowing Overlapping Boundaries in Source Code using a Search Based Approach to Concept Binding

Nicolas Gold Mark Harman Zheng Li
Kiarash Mahdavi
King's College London
Computer Science Department
Strand, London WC2R 2LS, England, United Kingdom

Abstract

One approach to supporting program comprehension involves binding concepts to source code. Previously proposed approaches to concept binding have enforced non-overlapping boundaries. However, real-world programs may contain overlapping concepts. This paper presents techniques to allow boundary overlap in the binding of concepts to source code. In order to allow boundaries to overlap, the concept binding problem is reformulated as a search problem.

It is shown that the search space of overlapping concept bindings is exponentially large, indicating the suitability of sampling-based search algorithms. Hill climbing and genetic algorithms are introduced for sampling the space. The paper reports on experiments that apply these algorithms to 21 COBOL II programs taken from the commercial financial services sector. The results show that the genetic algorithm produces significantly better solutions than both the hill climber and random search.

1. Introduction

Program comprehension is one of the most expensive activities in software maintenance and many tools and techniques have been created to reduce the time and expense involved [1, 3, 12, 19, 26]. Concept assignment techniques, such as Hypothesis-Based Concept Assignment (HB-CA) have been successfully employed to assign descriptive terms to source code as a means of supporting program comprehension [5, 8]. The concepts resulting from recent concept assignment techniques such as HB-CA are distinct, non-overlapping segments of code, which relate to computational intent.

However, complete, distinct localisation of concepts within code may be reliance upon too rigid an assumption within real program applications [21]. Concepts bound without this assumption may be a better representation of computational intent in the code. However, allowing overlap, results in an exponentially large search space, suitable for sampling based search algorithms. This paper uses hill climbing and genetic algorithms to search for concept bindings in code that are allowed to overlap.

```
MOVE  'EXAMPLE' TO PRINT-LL.
MOVE  POLICY-NUM TO OUT-PNUM.
MOVE  '13' TO PRINT-CC
MOVE  SCHEME-REF TO OUT-SERF.
CALL  'PRINT'  USING P-PRINTLINE
CALL  'WRITE' USING OUT-REC
```

Figure 1. Overlapping concepts example [6]

Figure 1 contains an example of a concept overlap in a code fragment of COBOL II [6]. In this example the first, third and fifth lines indicate a **Print** concept and the second, fourth and sixth lines indicate a **Write** concept. It is impossible to determine where one concept ends and the other starts. It could also be reasoned that the last two lines indicate a **Call** concept.

HB-CA is a plausible reasoning technique with a linear growth in computation cost that merits its application for large program studies [2]. This made it a suitable candidate for studying a large number of programs of varying size in this experiment. In addition, the concept binding mechanism (explained further in section 2) could be adapted easily as a means to drive the concept binding process for the new search based techniques.

This paper contains the details of an investigation into the use of search based approaches for concept assignment that allow overlapping concept bindings. Genetic algorithms (GA) and hill climbing (HC) search algorithms were used. HC was selected to determine if solutions achieved by using a "quick and simple" local search were able to compete with the more computationally intensive results from the GA. The results demonstrate the more focused concepts created due to the use of overlapping concept assignment with the GA and HC over the original HB-CA algorithm.

The overall contributions of this paper can be summarised as:

- The concept binding problem is reformulated as a search problem to allow overlapping concept boundaries.
- Genetic and hill climbing algorithms are used to search for solutions to this problem.
- The results of experiments with an implementation of these algorithms on 21 commercial Cobol II programs indicate that GA produces significantly better results than either hill climbing or random search.

2. Hypothesis-Based Concept Assignment

This section contains a brief explanation of the HB-CA algorithm [5, 8], upon which the results of this paper are based.

HB-CA requires a library or knowledge base. This library is a semantic network, composed of *concepts* and *indicators*. Indicators are evidence for concepts within the implementation. Concepts are the terms nominated by the user to describe items or activities in the domain. The library also includes relationships between concepts, used to identify composite or specialised concept binding. A concept may take the form of an Action or Object. Action concepts carry out operations, for example **Write** is an Action concept. Object concepts are concepts that can be acted upon by Action concepts and their presence together may suggest the existence of a composite concept, for instance the Object concept **File** and **Write** can create the composite **Write File** concept. The more general forms of object concepts are regarded as primary while the more specialised form are regarded as secondary. Composite concept may be created by using identified relationships within the library between Action and Object/Specialised concepts. The HB-CA algorithm is summarised in three stages, *hypothesis generation*, *segmentation* and *concept binding*.

2.1. Hypothesis Generation

The *hypothesis generation* stage draws on source code as input. The library is used whilst scanning the source code for indicators of various concepts. For each matching concept, a hypothesis is generated and stored. The list of hypotheses is ordered according to the position of the indicators in the source code. The ordered list of hypotheses, called the hypothesis list is then used for segmentation and concept binding. Figure 2 contains an example of a generated hypothesis list. The created hypothesis list forms the input for the search based algorithms.

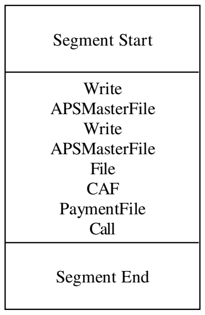

Figure 2. Example of a generated hypothesis list

2.2. Segmentation

The *segmentation* stage attempts to create distinct, disjoint segments within a hard segment. Hard segments are natural segment boundaries such as procedure divisions. A Self Organising Map [17] creates segments of high conceptual focus according to the distribution of the Action concepts within a hard segment.

2.3. Concept Binding

The *concept binding* stage is carried out by the Concept Assigner. The Concept Assigner evaluates each segment in term of concept occurrence. The library is used to determine the possible composite concepts within each segment. The strength of evidence for a concept is equivalent to the number of hypotheses that could indicate the presence of that concept. The Concept Assigner also requires the presence of at least one action concept in addition to a user defined minimum number of hypotheses (minimum evidence) to create a *concept binding*. Assuming these conditions are satisfied, the concept with the strongest evidence will be selected as the winner. A set of disambiguation rules is applied to select a winner in case of ties [8]. The segments are then bound to the winning concepts and highlighted in code.

3. Defining the Search Problem

A search problem is the algorithmic identification of a solution from a solution space. As discussed in Subsection 2.1, the input for this search algorithm is the hypothesis list generated by the HB-CA algorithm. Therefore the problem can be defined as searching for segments of hypotheses in each hypothesis list according to predetermined fitness criteria such that each segment has the following attributes:

- Each segment contains 1 or more neighbouring hypotheses.
- There are no duplicate segments.

The search fitness criteria's aims are twofold:

1. Guide the search to finding segments of strongest evidence.
2. Binding as many of the hypotheses within the hypothesis list without compromising the segments strength of evidence.

3.1. Size of the Search Landscape

In this section the size of the search (number of possible solutions) and its growth according to the number of hypotheses within the hypothesis list are analysed. The number of possible segments which can be created within a hypothesis list, according to the definition in Section 3, is given explicitly by the following:

$$s = \frac{h(h+1)}{2}$$

Where s is the number of segments and h is the number of hypotheses. The size of the search (the number of possible solutions) is the number of possible combinations of these segments. That is, the number of subsets of the set of all possible hypotheses, not including the empty set. This search space size is:

$$c = 2^s - 1$$

Therefore the size of the search space is exponential in the number of hypotheses. The exponential increase in the search space favours a sampling-based search approach such as the GA and HC approaches considered in this paper.

3.2. Fitness Function

For the fitness function to effectively guide the search, it must be able to evaluate each solution according to the strength of evidence and hypothesis list coverage. The first step involves the recognition of the strongest concept within each segment of a particular solution. This is achieved by following the same process as the HB-CA's *concept binding* method previously discussed in Section 2.3.

The overall fitness is then evaluated to find the segmentation strength in addition to the hypothesis list coverage. The segmentation strength is the combination of inner fitness and the potential fitness of each segment. The inner fitness of each segment is assessed as:

$$fit_i = signal_i - noise_i$$

Where fit_i signifies the inner segment fitness, $signal_i$ represents the signal level i.e. number of hypotheses within the segment that contribute to the winner and $noise_i$ represents the noise level i.e. the number of hypotheses within the segment that do not contribute to the winner. The inner segment fitness results in recognition of higher fitness for segments with more evidence indicating their winning concept. In addition each segment is evaluated with respect to the entire segment hypothesis list:

$$fit_p = signal_i - signal_p$$

The potential segment fitness, fit_p is evaluated by taking account of $signal_p$, the number of hypotheses outside of the segment that could have contributed to the segment's winning concept if they were included in the segment. This facet of the segment fitness effectively guides the search towards the creation of larger segments, incorporating as much of the signal as possible. The overall segment fitness is evaluated by combining the inner and potential segment fitness into an overall segment fitness:

$$segfit = fit_i + fit_p$$

The overall segment fitness ($segfit$) attempts to guide the search towards larger segments of high signal and low noise level. Finally the total segment fitness is calculated as:

$$totsegfit = \sum_{s=1}^{n} segfit(s)$$

where n is the number of segments within the solution.

Hypothesis list coverage is the second facet of the fitness calculation. Increased coverage of the hypothesis list results in further coverage of the original program code, which could potentially improve program comprehension. Hypothesis list Coverage is defined as:

$$h_c = h - h_n$$

where h is the number of hypotheses within the hypothesis list and h_n the number of hypotheses not covered by any segments within the solution. For a fair comparison of solutions, given that solutions may have a varying number of segments and coverage, a normalised version of the fitness is evaluated:

$$Fitness = \frac{totsegfit + h_c}{2totseglength + h}$$

where

$$totseglength = \sum_{s=1}^{n} seglength(s)$$

and $seglength(s)$ is the number of hypotheses in segment s.

3.3. Genetic Algorithms

GAs are a collection of heuristic population-based evolutionary search techniques. Traditionally individual solutions with a GA population are also referred to as chromosomes and their constituent parts as genes [14]. The genes represent a coding of the search parameters rather than directly used parameters. GA search starts with a random population of potential solutions. It then employs evolutionary inspired operators whilst guided by a fitness function to evolve fitter solutions. Amongst the many subtly varied definitions of a GA the following mechanisms are generally agreed upon [10, 18, 23]:

- Selection: The probability based sampling of the current population of solutions, guided by a fitness function. The selected solutions participate in the creation of the next generation of solutions using GA search operators.
- Crossover: The primary search operator. Involves the recombination of pairs of selected solutions to create offspring for the next generation of solutions.
- Mutation: The secondary search operator. Involves the random selection and change of genes in a newly created population with the aim of reducing population stagnation.

Selection, crossover and mutation are used to create subsequent generations of populations until a Stopping Condition is satisfied. The Stopping Condition maybe defined in a variety of ways. Example Stopping Condition could be based on the number of generated new populations or the level of increase in the population fitness over generated populations.

Population size, selection, crossover and mutation are governed by a set of heuristics. Crossover rate determines the probability of recombination for a pair of selected individuals. Selected individuals not recombined due to crossover rate are copied directly into the new population.

Tournament selection is explained in this section as it is used as the crossover operator for the proposed GA. Initially a random pair of chromosomes are selected for tournament selection. The chromosome with the higher fitness may then be selected to participate in crossover and mutation according to the tournament coefficient heuristic. The tournament coefficient is the probability of the fitter chromosome being selected and is usually set strongly in favour of the fitter chromosome. Mutation rate determines the probability of a mutation per gene(bit) of a solution of the new population. This probability is normally set to be very low to avoid the search from deteriorating into a random search. The crossover and mutation mechanisms are also dependent on the structure of the chromosome. The proposed structure used in this paper and its implications for these operations are discussed further in Section 3.5.

3.4. Hill Climbing Algorithms

Hill climbing is a local search technique. It starts from a single randomly-created solution. A set of mutations are then used to define a set of potentially fitter solutions. The set of solutions created by the mutation operator are referred to as the neighbouring solutions. The search at each stage selects a fitter neighbouring solution and examines further neighbouring solutions from the newly discovered solution. The number of considered neighbouring solutions is a heuristic that determines the minimum number of examined before a selection decision is made. Due to the local nature of the search, it is possible for the starting solution to terminate at a locally optimal solution of relatively low fitness. The search attempts to reduce this possibility by repeatedly restarting from a new random solution or by using the characteristics of the current solution to create a new solution. The search ends when no fitter neighbours can be found and the Sopping Condition has been reached. The Stopping Condition search may be determined heuristically by limiting the number of allowed restarts or algorithmically when no fitter solution for a restart can be found by using the restart mechanism. The proposed HC for this experiment uses characteristics of the final solution for the restart operation and stops when it cannot create better solutions using restart. The restart operation and permitted mutations are dependent on the defined structure of the solution.

3.5. Solution Structure

The scope of resulting solutions are affected by what constitutes a solution in the HC and chromosomes in the GA population. In this case, each solution needs to represent all possible overlapping and not-overlapping segment allocations. The proposed solution defines a segment as a pair of values where each value represents the location of start and end hypothesis within the ordered hypothesis list. Since the number of segments within a hypothesis list in not predetermined, the length of a potential solution can also vary. A messy GA chromosome structure was chosen as a suitable representation [11] since it allows the chromosome to have

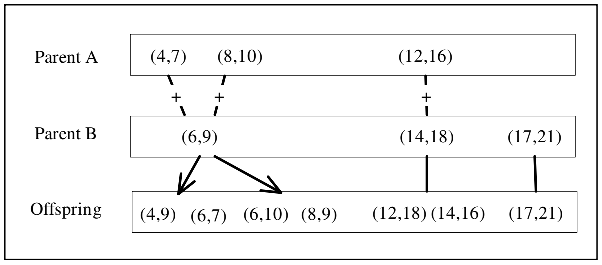

Figure 3. Example crossover operation on GA chromosomes

variable length. In the proposed solution, a chromosome is made up of a set of one or more segment representations, referred to as segment pairs.

A difficulty that was detected during implementation of this representation is the potential for an unmanageable increase in the size of solutions. Although the problem definition as explained in Section 3 results in the elimination of all duplicates segment pairs, this never the less leaves a potentially large number of segment pairs. The fitness function was exploited in this case to reduce this size even further. In the proposed solution all segments with the same winning concept that overlap are compared and all but the fittest segment are removed from the solution.

The crossover implemented utilises the location of the segment pairs, where only segment pairs of overlapping locations are recombined and the remaining are copied to the new chromosome. Figure 3 contains an example of the recombination process on two chromosomes and the resulting offspring.

The GA mutation operation on the proposed solution structure is different from HC mutation. The GA mutation operator is a secondary search operator, primarily concerned with population stagnation. Therefore the mutation operator can randomly replace any hypothesis location within any segment pair with any other valid hypothesis location with the concern for causing the search to become overly randomised. As a result this mutation model is used for the GA, where a mutation can occur on each hypothesis location according to the mutation rate, where a mutation results in the replacement of a hypothesis location value in a segment pair with a random and valid hypothesis location value. Conversely such mutations would cause a local search technique such as hill climbing to become akin to a random search. To reduce this effect, the proposed HC mutation operator generates new solutions by selecting a segment pair and increasing or decreasing one of the location values by a single increment. The resulting mutations are a set of similar neighbouring solutions that can be used to describe the local search landscape of the hill climbing algorithm. Finally the proposed HC takes advantage of the proposed GA crossover operation GA for the Restart mechanism as discussed in Section 3.4. This entails recombination of all segment pairs in order to create new segments pairs, which are then added to the current Solution if their inclusion results in an improvement to the fitness value.

Name	Purpose	LoC
GD95	Reformat master file	89
GB92/6	Print covering letters	190
GD25	Print payments due list	238
GD12	Print payment file	285
GD30	Print cheques and statements	337
GD60	Products/vestings total	387
GD91	Print entire CAF	441
GD96	File verification	491
GD83	Extract school fees payments	547
GB64	Letter print	596
GD26	Print annuity statement	650
GD81	Validate school fees data entry	701
GB73	Create payment file	728
GD28	Produce payment schedule	807
GD67	Update annuity file	879
GB01	Record update/reporting	1013
GD02	CICS enquiry screen	1117
GB07	File totals	1162
GB03	Policy file update	1237
GBCM0133	Compute control totals	1310
GB08	Identify outstanding payments	1374

Table 1. List of used programs, their purpose and size.

4. Empirical Study

An empirical study was carried out to identify the best of the proposed algorithms for concept assignment that allow overlapping concept boundaries based on the proposed fitness discussed in Section 3.2. 21 COBOL II programs from the commercial financial services sector were included in the study. A list of these programs, their purpose and size in lines of code (LoC) is in Table 1. A set of hypothesis lists were generated using the HB-CA hypothesis generator discussed in Section 2.1. The generated hypothesis lists have a size ranging from 1 to 191 hypotheses. Due to the probabilities involved in the generation of the initial population for the GA and initial solution for the HC and the inherent randomness involved in the search operators, it is possible that a single execution of the algorithm may produce uncharacteristic results. In order to better evaluate characteristic solutions, 10 GA and HC runs were carried out per hypothesis list.

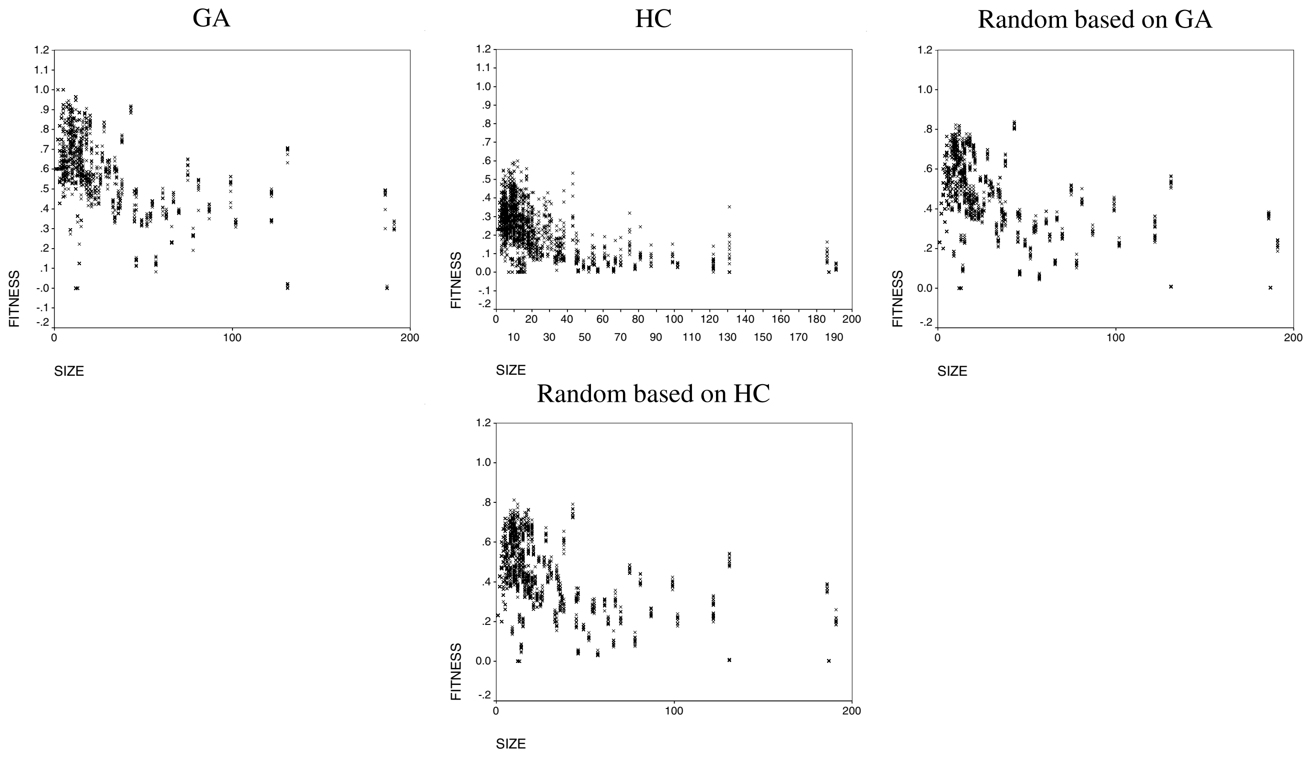

Figure 4. Scatter graphs of fitness against hard segment size.

The set of heuristic values for the GA and HC were derived by trials and experimentation on some of the smaller hypothesis list. The GA population consists of 100 chromosomes, which are created randomly for the initial population. This involves the creation of a random number of genes or segment pairs. The number of genes is set between a minimum of 5 to a maximum based on the number of hypotheses in the hypothesis list. The tournament selection's coefficient was set to 0.99 and crossover and mutation rates were set to 0.8 and 0.001 respectively. The GA search terminates when the average fitness of the population does not change over a sequence of 50 generations. The initial solutions used for the HC algorithm are produced by following the GA initial chromosome creation mechanism. The HC moves to a new solution every time a fitter solution is discovered. The HC stopping condition is met when no fitter neighbouring solutions are present and no fitter solution can be achieved by using the restart mechanism explained in Section 3.5.

The GA and HC results were also compared to sets of randomly generated solutions for each hypothesis list. These solutions were created according to the solution structure described in Section 3.5. The number of generated random solutions for each hypothesis list was equal to the number of evaluated solutions by the GA and HC algorithms for each hypothesis list. Comparison of the GA, HC and random results are discussed in Section 4.1.

Traditional non search-based HB-CA was also used for concept assignment on the 21 COBOL II programs to generate a set of solutions without overlapping concept boundaries. The minimum evidence level for concept binding was set at 3 hypotheses (details of HB-CA concept binding and evidence level have been discussed in Section 2.3). Analysis was carried out on the results from the proposed GA and HC algorithms and the HB-CA results to find the best algorithm. Due to the different HB-CA search criteria, the comparison was based on a different measure. This measure along with the results of the HB-CA comparison are discussed in Section 4.2.

The results are presented as scatter graphs and boxplots. The scatter graph's vertical axis represents the fitness value and its horizontal axis represents increasing hypothesis list size. Similarly the boxplot's vertical axis represents fitness values. However to reduce clutter in the presentation of boxplots, caused by the large variety in hypothesis list sizes, it was necessary for the boxplots to be drawn against increasing ranges of hypothesis list size. The 5 increasing ranges used are 1 to 38, 39 to 76, 77 to 114, 115 to 152 and 153 to 191. For example the range 1 to 38 represents all the fitness values resulting from hypothesis list sizes of between 1 to 38 (inclusive) hypotheses. Each boxplot represents the distribution of fitness values for a particular hy-

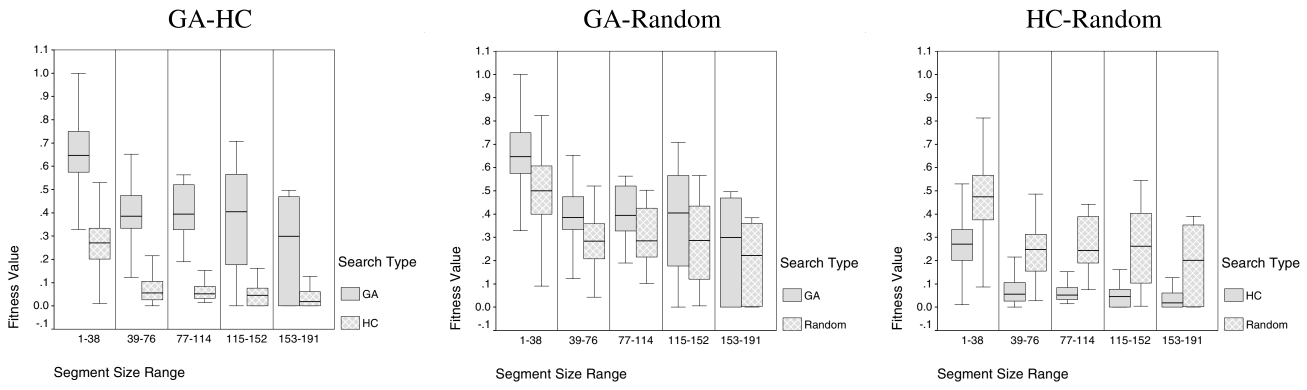

Figure 5. Boxplots of GA, HC and Random fitness results compared across increasing hypothesis List size range.

pothesis list size range. The length of the box corresponds to the interquartile range and contains 50% of the cases. The line across the inside of the box indicates the median value. The protruding lines (whiskers) represent the smallest and largest fitness values that are not outliers, where outliers are values which are 1.5 box-lengths away from the edge of the box.

4.1. GA, HC and Random Results

The results from the GA, HC and random experiment were compared based on their fitness values, which were calculated by the fitness function described in Section 3.2. Scatter graphs labeled GA, HC, random based on GA and random based on HC in Figure 4 contain the fitness distribution ordered by hypothesis list size for GA, HC and GA and HC random search respectively. The distribution of GA fitness results according to Figure 4 is similar to the random and HC distributions but with a shift towards higher fitness values. On the other hand, the HC results are clearly of inferior fitness to all other results presented by the scatter graphs in Figure 4. The boxplot of fitness values for paired algorithms against increasing hypothesis list size are overlaid in Figure 5. The superior GA results compared to HC and random is also conveyed by this Figure. The GA-HC and HC-random graphs in this Figure once again highlight the inferior HC fitness results compared to the other results. Not surprisingly the results also demonstrate the increasing difficulty for all search algorithms as the hypothesis list size increases. This observation corresponds to the rapid increase in the search space discussed in Section 3.1. The HC also has the smallest distribution of results compared to the GA and random results. The implications of this observation are further discussed in Section 6.

Pair-wise comparison of the GA, HC and random by the *Wilcoxen Signed Rank Test* was used to ascertain the statistical significance in the observed strength of GA and the weakness of the HC results in terms of fitness values. The *Wilcoxen Signed Rank Test* reported the level of significance to be less that 0.0005 when comparing GA results against HC and GA results against randomly created results. This level of significance represents a statistically significant improvement to the fitness for the GA compared to the HC and randomly generated solutions. The *Wilcoxen Signed Rank Test* reported a level of significance of below 0.0005 meaning a statistically significant worsening of HC results compared to the randomly generated solutions.

4.2. GA, HC and HB-CA Results

As described in Section 2, The HB-CA algorithm carries out segmentation based on a different set of criteria to GA and HC algorithms. For a more impartial comparison between these algorithms, the results are evaluated based on the signal to size ratio, where the signal represents the number of hypotheses within a segment that contribute to the winning concept and Size represents the number of hypotheses within that segment.

The scatter graphs in Figure 6 display the distribution of signal to size ratios of created segments across increasing hypothesis list size for the GA, HC and HB-CA algorithms. Most noticeable from these graphs is the lack of solutions with low signal to size ratios for the scatter graphs of GA and HC when compared to HB-CA. The GA-HBCA boxplots in Figure 7 further illustrate characteristically better signal to size ratios achieved by the GA algorithm compared to HB-CA across the range of of hypothesis lists. The HC results in the HC-HBCA boxplots in Figure 7, although not as clearly improved as the GA results, are generally better when compared to HB-CA. The graphs in Figure 7 also display consistently higher GA results in comparison to HC, identifying the GA as the best overall algorithm based on

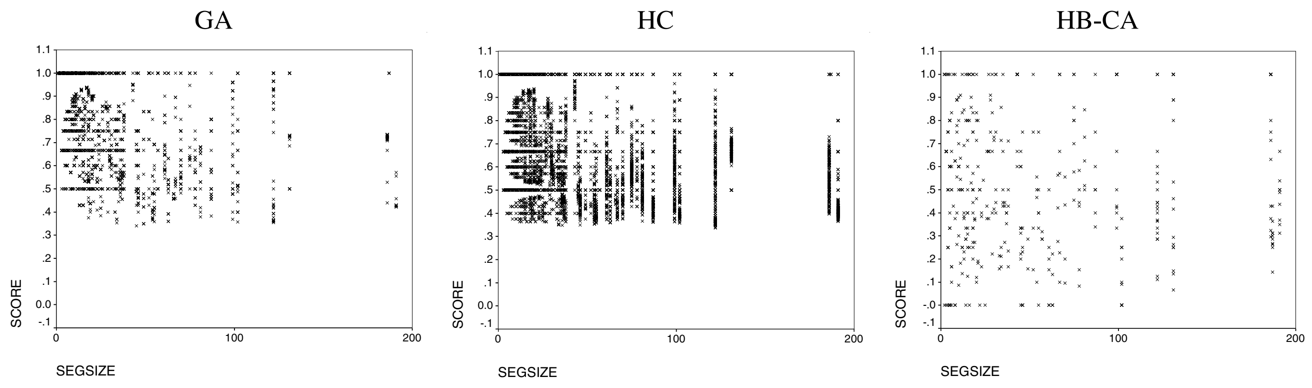

Figure 6. Scatter graphs of signal to size ratio against hard segment size.

the signal to size ratio.

Pair-wise comparison of the GA, HC and HB-CA was carried out by using the *Wilcoxen Signed Rank Test* to determine if the strength of GA and HC results against HB-CA were significant in terms of signal to size ratio. The test reported a significant difference of less than 0.0005 for all of these comparisons. This implies the GA and HC signal to size ratios were significantly better than HB-CA. Further *Wilcoxen Signed Rank Test* between GA and HC also yielded a significance difference of below 0.0005, meaning in terms of signal to size ratio, the GA results were also significantly better than the HC.

5. Related Work

Concept assignment has been defined by Biggerstaff as "...a process of recognising concepts within a computer program and building an 'understanding' of the program by relating recognised concepts to portions of the program, its operational context and to one other [2]."

The two major research issues of concept assignment have been identifies by Tilly as *segmentation* and *concept binding* [25]. Segmentation is the process of grouping pieces of conceptual information generated from the source code. Concept binding involves the analysis of segments in order to determine the most plausible concept assignment for each segment [7]. The segmentation and concept binding process are intricately and naturally linked. The location and size of the segment determines the assigned concept. The strength of the assigned concept on the other hand determines the quality of segmentation.

Concept assignment techniques are carried out by intelligent agent tools. They traditionally fall within the following categories [2].

1. Domain specific, rule based, model driven systems that answer specific questions. These depend on manually created databases which describe the software system. This approach is exemplified by the LaSSIE System [4].
2. Plan driven, algorithmic and based on a precise set of understanding and recognition rules. Examples of this method are Programmer's Apprentice [24] and GRASPR [28].
3. Model driven systems which use plausible reasoning. Examples of this technique are DM-TAO [2], IRENE [16] and HB-CA [5, 8].

The tools that employ approaches in first and second category are capable of completely deriving concepts within small scale programs but due to their overwhelming computational cost are not suitable for larger-scale programs [2, 8]. Conversely, the third approach has a linear computational growth in program size, but suffers from imprecision in results [2, 8].

HB-CA plausible reasoning technique has recently been proposed as a means for more complex reveres engineering and software testing [9, 13]. This involves the use of program slicing [27] in conjunction with HB-CA derived concepts to create executable concept slices (ECS) [9, 13]. ECS involves the slicing of concept bindings from the HB-CA according to the system dependence graph approach of Horwitz et al [15]. The resulting ECS are proposed to posses the higher level abstraction of concepts alongside the useful executability of a program slice [9, 13].

Another recent related technique involves the use of Latent Semantic Analysis(LSA) for concept location [22]. According to Landauer et al. "LSA is a fully automatic mathematical/statistical technique for extracting and inferring relations of expected contextual usage of words in passages of discourse" [20]. The technique involves the analysis of user queries alongside the parsing and analysis of code as text to identify concepts. One of the strength of this technique is its independence from the programming language being analysed. Current comparison with related methods shows

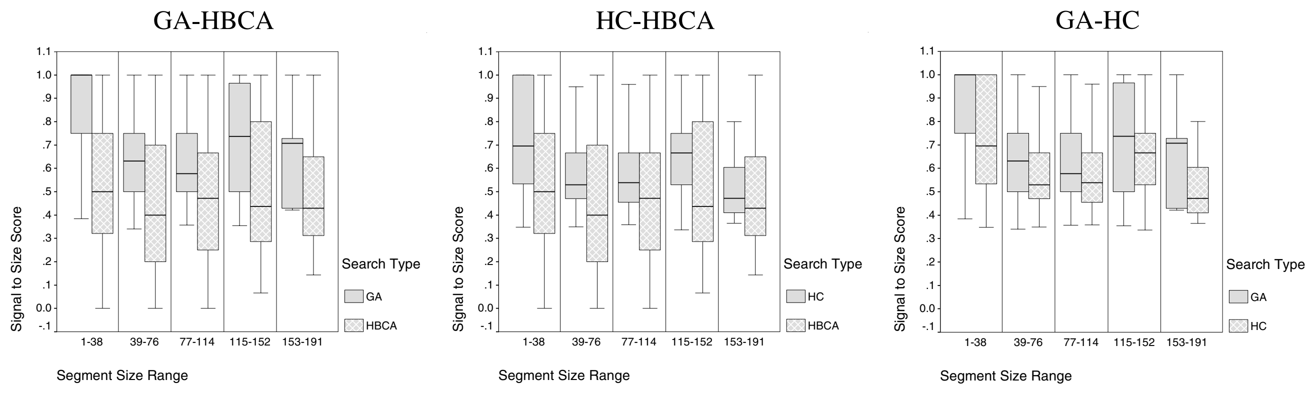

Figure 7. Boxplots of HB-CA, GA and HC signal to size ratio compared across increasing hypothesis list size range.

this technique to be easy and flexible whilst able to provide good results [22].

6. Conclusions and Future Work

An approach to permit overlapping concept boundaries for concept assignment was presented in this paper. The problem was defined, analysed and formulated as a search problem in terms of search space, fitness function, GA and HC algorithms and solution structure in Section 3. An empirical study was carried out in two parts to determine the best algorithm in Section 4. First study compared the proposed GA, HC and randomly generated solutions based on the proposed fitness function. The second study compared the GA and HC results with HB-CA, based on signal to size ratio. The results of these studies were discussed in Sections 4.1 and 4.2 respectively.

The GA results produced significantly stronger fitness values according to the proposed fitness function. In addition, the GA results were significantly better than HC and HB-CA results according to the signal to size ratios, as discussed in Section 4.2. This identified the GA as the best of the proposed algorithms for concept assignment which allow overlapping concept boundaries. On the other hand the HC results were somewhat disappointing as they were found to be significantly worst than GA and randomly generated solutions based on the proposed fitness function.

However HC produced stronger results when compared to the HB-CA on the signal to size measure. One possible explanation for this behaviour may be the increase in complexity of the search due to the inclusion of hypothesis list Coverage as part of the fitness criteria, where a local search algorithm such as hill climbing is simply not adequate. Another explanation could be the inadequacy of the current neighbourhood definition and the need for examining alternative neighbourhood definitions.

Another observation made was on the comparatively small range of HC fitness values in Section 4.1 compared to other search algorithms. The smaller fitness distribution implies that a set of similar fitness values were achieved by the HC from random starting points, which in turn may indicate a large number of similar locally optimum solutions within the search space. Since the shape of the landscape is directly effected by the neighbourhood definition, this observation also strengthens the need for more suitable neighbourhood definitions for the HC algorithm.

Further research is required to analyse the resulting concept bindings as reflected in code. Useful future investigations may take the form of measuring the level of agreement for the location of concept bindings, analysis of the size and distributions of created segments in the hypothesis list and the size and distribution of the resulting concept bindings in program code. These investigation may also help to demonstrate the potential offered in program comprehension by the proposed techniques and whether the inclusion, extent and frequency of overlap could help or hinder program comprehension.

7. Acknowledgements

This work is supported by EPSRC Grants GR/T22872/ and GR/R71733. We would also like to gratefully acknowledge the support of the Computer Science Corporation. The corresponding author for this work is Dr. Kiarash Mahdavi (email: kiarash.mahdavi@kcl.ac.uk). The authors in all publications relating to the CONTRACTS project, including this publication, are ordered alphabetically.

References

[1] I. D. Baxter. Design maintenance systems. *Communications of the Association for Computing Machinery*, 35:73–

89, April 1992.

[2] T. J. Biggerstaff, B. G. Mittbander, and D. Webster. The concept assignment problem in program understanding. In *Proceedings of the 15th international conference on Software Engineering (ICSE 1993)*. IEEE Computer, 1993.

[3] J. R. Cordy, T. R. Dean, and A. J. Malton. Source Transformation in Software Engineering using the TXL Transformation System. *Information and Software Technology*, 44:827–837, October 1996.

[4] P. Devanbu, R. Brachman, P. Selfridge, and B. Ballard. LaSSIE: A Knowledge-Based Software Information System. *CACM*, 34(5):34–49, May 1991.

[5] N. Gold. Hypothesis-Based Concept Assignment to Support Software Maintenance. In *Proceedings IEEE International Conference on Software Maintenance 2001*, pages 545–548, 2001.

[6] N. Gold and K. H. Bennett. A Flexible Method for Segmentation in Concept Assignmen. In *IWPC*, pages 135–144, 2001.

[7] N. Gold and K. H. Bennett. Hypothesis-based concept assignment in software maintenance. *IEE Proceedings - Software*, 149(4):103–110, 2002.

[8] N. E. Gold. *Hypothesis-Based Concept Assignment to Support Software Maintenance*. PhD thesis, Research Institute in Software Evolution, Department of Computer Science, University of Durham, 2000.

[9] N. E. Gold, M. Harman, D. Binkley, and R. M. Hierons. Unifying program slicing and concept assignment for higher-level executable source code extraction. *j-SPE*, 35(10):977–1006, Aug. 2005.

[10] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[11] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[12] GrammaTech-CodeSurfer.
http://www.grammatech.com/products/codesurfer/.

[13] M. Harman, N. Gold, R. M. Hierons, and D. Binkley. Code Extraction Algorithms which Unify Slicing and Concept Assignment. In *WCRE*, pages 11–21, 2002.

[14] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press (Second edition MIT Press 1992), second edition, 1975.

[15] S. Horwitz, T. W. Reps, and D. Binkley. Interprocedural Slicing Using Dependence Graphs. In *PLDI*, pages 35–46, 1988.

[16] V. Karakostas. Intelligent Search and Acquisition of Business Knowledge from Programs. *Journal of Software Maintenance: Research and Practice*, 4:1–17, 1992.

[17] T. Kohonen. *Self-Organising Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, third edition edition, 2001.

[18] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms. Generation, Enumeration and Search*. CRC Press, 1999.

[19] B. Laguë and M. Dagenais. An Analysis Framework for Understanding Layered Software Architectures. In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, pages 37–44. IEEE Computer Society, June 1998.

[20] T. K. Landauer, P. W. Foltz, and D. Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, 25:259–284, 1998.

[21] S. Letovsky and E. Soloway. Delocalized Plans and Program Comprehension. *IEEE Software*, 3(3):41–49, May 1986.

[22] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An Information Retrieval Approach to Concept Location in Source Code. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 214–223, Nov. 2004.

[23] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 7 edition, 2001.

[24] C. Rich and R. C. Waters. The programmer's apprentice. *Computer*, 21(11):10–25, Nov. 1988.

[25] S. Tilley. A Reverse-Engineering Environment FrameWork. Technical Report CMU/SEI-98-TR-005, ESC-TR-98-005, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, April 1998.

[26] M. Ward. *Proving Program Refinements and Transformations*. PhD thesis, Oxford University, 1989.

[27] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10:352–357, July 1984.

[28] L. Wills. *Automated Program Recognition by Graph Parsing*. PhD thesis, MIT Artificial Intelligence Lab, July 1992.