Late Acceptance Hill Climbing Based Strategy for Test Redundancy Reduction and Prioritization

*Rohani Bakar [#1], Kamal Z. Zamli [#2], and Basem Al-Kazemi[#3]

[#1]*Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, 26300 Kuantan, Pahang, Malaysia*

[#2]*College of Computer and Information Systems, Umm Al-Qura University, Makkah, Kingdom of Saudi Arabia*
*Corresponding Author: rohani@ump.edu.my

Abstract— **Software testing relates to the process of accessing the functionality of a program. To ensure conformance, test engineers often generate a set of test cases to validate against the user requirements. When dealing with large line of codes (LOCs), there are potentially issues of redundancy as new test cases may be added and old test cases may be deleted during the whole testing process. To address redundancy issues, many useful strategies (e.g. HGS, GE, and GRE) have been developed in the literature. These strategies often put focus on getting the most minimum test suite size but give poor emphasis on test prioritization (i.e. ordering of tests). Here, as most testing activities happen toward the end of software development, testers are often forced to consider partial test suite, that is, to be in line with the project deadline. In this manner, some impactful defects may be missed owing to the need to accommodate deadline shift from earlier development activities. In order to address these issues, this paper highlights our on-going work on the development of a novel test redundancy reduction strategy based Late Acceptance Hill Climbing, called (LAHCS). LAHCS is the first known strategy that adopts Late Acceptance Hill Climbing Algorithm for test redundancy reduction and prioritization.**

I. INTRODUCTION

To ensure quality and conformance, developers often rely on testing to reveal defects. Typically, testing is guided by the designed test suite made of a set of test cases. These test cases are usually backward traceable to the corresponding design, product requirements and right through the stakeholder's justification. Generally, test cases are dynamic entity. Owing to the need to address defects and accommodate stakeholders' change requests during the development process, new test cases may be added whilst existing test cases be updated or be removed completely. For these reasons, there is potentially significant probability for test redundancy, that is, one requirement is covered by more than one test case. Although desirable in some class of systems, test redundancy often incurs unnecessary costs.

In the literature, test redundancy issues have been addressed by many researchers resulting into many helpful strategies (e.g. HGS [1], GE[2], and GRE[3]) Although useful in term of systematically sampling of the appropriate test case for consideration, existing strategies have not sufficiently dealt with test prioritization. As most testing activities happen towards the end of software development, testers are often forced to prioritize and consider partial of the test cases, that is, to be in line with the project deadline. Addressing the aforementioned issues, this paper describes a novel approach of adopting Late Acceptance Hill Climbing (LAHCS) based Strategy for test redundancy reduction and prioritization. LAHCS serves as our research vehicle to investigate the effectiveness of Late Acceptance Hill Climbing Algorithm for test redundancy reduction and prioritization.

The rest of the paper is organized as follows. Section II gives an overview of the test redundancy reduction and prioritization problem and highlights the related work. Section III describes our strategy within the Late Acceptance Hill Climbing Algorithm. Section IV highlights our benchmark against other strategies. Finally, Section V gives our conclusion and future work.

II. OVERVIEW AND RELATED WORK

Test redundancy reduction and prioritization problem can be viewed as a set covering problem as follows [4]:

Given: A test suite TS, a list of testing requirements $r_1, r_2, \ldots r_n$ with well-defined prioritization contribution, that must be tested to provide the desired testing coverage of the program, and a list of subsets of TS, T_1, $T_2, \ldots$ T_n, one associated with each of the r_i's such that any one of the test cases t_j belonging to T_i can be used to test the requirement r_i.

Problem: Find an ordered representative set of test cases t_j according to defined priority that will satisfy all of the r_i's.

Many useful strategies have been developed to address the aforementioned problem in the last 20 years. Perhaps, the pioneer work on test redundancy reduction is based on that of Chavatal[4]. He introduces a novel strategy based on the greedy heuristics. Initially, the Chavatal's strategy greedily picks a test case t_i that covers the most requirements. Then, all the requirements that are covered by t_i are marked. The

whole cycle is repeated until all requirements are marked. Although helpful, Chavatal's strategy appears not optimal and does not deal with prioritization.

Complementing Chavatal's work, Harrold et al develops a similar strategy, called HGS [1]. Unlike Chavatal's strategy, HGS greedily ranks the cardinality of each requirement with the corresponding test case (from low to high) as the main basis for reduction. A requirement *A* has lower cardinality than a requirement *B* if *A* is covered by fewer test cases than *B*. Briefly, HGS works as follows. Initially, all covered requirements are considered unmarked. For each requirement that is exercised by one test case (i.e. cardinality of 1), HGS adds the test case into the minimized test suite and marks the covered requirements accordingly. Next, HGS considers the unmarked requirements in increasing order of cardinality of the set of test cases exercising each requirement. Then, HGS chooses the test case that would cover the greatest number of unmarked requirements associated with the current cardinality of interest. When there is a tie amongst cardinality of multiple test cases, HGS breaks the tie in favour of the test case that would mark the greatest number of unmarked requirements with the case sets of successively higher cardinalities. If the highest cardinality is reached, and the tie is not resolved, HGS arbitrarily selects one amongst those tied test case. Then, HGS marks the requirements covered by the selected test case. The whole iteration is repeated until all the requirements are completely marked. The main strength of HGS is the fact that it creates a subtle (and stable) prioritization of test cases during its selection process (i.e. based on cardinality). Here, hard to cover requirement with low cardinality are considered first and followed by other requirements in order of increasing cardinality. The main limitation of this approach is the fact that, in real testing endeavour, prioritization is not solely a function of cardinality. In fact, prioritization can also be a function of likelihood of faults as well as their impacts.

Lau and Chen introduce another variant greedy strategy, called GE [2]. In GE, the concept of concept of essential test case is introduced for the greedy selection of test cases. Here, essential test cases, $t_{essential}$, represent those test cases that when removed, some test requirements can never be satisfied. In a nut shell, GE works as follows. Firstly, GE selects the essential test cases $t_{essential}$ that cover the most uncovered requirements. Secondly, GE removes all the requirements covered by the chosen essential test cases $t_{essential}$. The process continues for all other essential test cases until completion. If there are any uncovered requirements, the GE iterative process will continue to greedily select test cases t_i that covers the most uncovered requirements much like Chavatal's approach [4]. Then, all the requirements covered by t_i are removed. The process is repeated until all requirements are covered. Implementation wise, GE is straightforward to implement as compared to HGS. Furthermore, as GE considers $t_{essential}$ before greedily selecting candidate test case, the test suite size offered by GE is at least the same of better than that of Chavatal. The same argument cannot be applicable when comparing HGS and GE. On the negative note, GE does not address prioritization issue.

As enhancement of GE, Chen and Lau later introduce the GRE strategy[3]. In addition to the concept of essential test cases, GRE also exploits the idea of redundant test case. In this case, if a test case satisfies only a subset of test-case requirements satisfied by another test case, then that particular test case is redundant. GRE starts by first removing redundant test cases from the test suite. In the process, GRE reduces the test suite and may make some test cases essential. Then, GRE applies the same algorithm as GE in order to choose the test cases that cover all the requirements. GRE inherit many advantages of GE. In fact, in the absence of redundant test case, GRE behaves much like GE. Interestingly, due to NP completeness of the test redundancy reduction problem, the performance of GE can still be better than GRE or even HGS in terms of test reduction. Similar to GE, GRE does not address the prioritization issue.

Shengwei et al adopts a strategy similar to GE [5]. Unlike GE, they exploits weighted set covering (for requirements) in order to eliminate test redundancy and prioritize the test suite according to cost order. The general performance of the algorithm appear the same to that of GE. On the negative note, although important, prioritization need not be considered merely on cost but on how effective of the tests being prioritized. As highlighted earlier, prioritization can also be a function of likelihood of faults as well as their impacts.

Galeebathullah and Indumathi develop a strategy that combines the set theory and greedy heuristics [6]. Initially, the strategy finds the intersection of each requirement with other requirements. If exist any intersection exist, the test cases are greedily combined and added to the final test suite. The process is repeated until all requirements are covered by the test case. In the work, prioritization issues are not reported. Additionally, no benchmarking result against other existing strategies is published.

Apart from the greedy heuristic approach, a number of researchers (e.g. Tallam and Gupta [7] and Ng et al [8]) have started to adopt the Formal Concept Analysis (FCA). Basically, FCA is a technique for classifying objects based upon the overlap among their attributes. For reduction, test cases are considered as objects and requirements as attributes. Relationship between objects and attributes corresponds to the coverage information of test case. Using concept analysis, maximum grouping of objects and attributes can be deduced (termed context) in a table. Here, facilitated by graphical concept lattice and based on the object and attribute reduction rules, objects (i.e. test cases) can be systematically reduced. Although helpful, FCA suffers from the problem of scale – when the formal objects and their attributes grew, it is almost impossible to construct and manipulate the concept lattice graphically. Hence, the applications of FCA for large scale test reduction (and prioritization) can be problematic and difficult.

In light of some of the problems highlighted earlier, this paper proposes the use of Artificial Intelligence Algorithm for test redundancy and prioritization problem. Specifically, this paper adopts a new variant of Hill Climbing Algorithm, termed the Late Acceptance Hill Climbing Algorithm [9-11].

The main feature of Late Acceptance Hill Climbing Algorithm is the fact that it provides significant improvements over its predecessor in terms of performance (and still maintains the Hill Climbing simplicity). Unlike the original Hill Climbing, Late Acceptance Hill Climbing algorithm allows worsening moves. In this manner, the iterative search in Late Acceptance Hill Climbing can be prolonged to avoid the local minima problem inherent to the original Hill Climbing algorithm. Another useful feature of the Late Acceptance Hill Climbing algorithm is that it has a single parameter for manipulation, hence, not vulnerable to inadequate parameterization and insufficient tuning [9].

III. Late Acceptance Hill Climbing Algorithm For Test Redundancy Reduction and Prioritization

In a nut shell, Late Acceptance Hill Climbing (LAHC) adopts an iterative neighbourhood search process similar to its predecessor. Nonetheless, unlike its predecessor which compares the candidate solution with the current one for acceptance (i.e. when the cost function is not worse), LAHC delays this comparison with a solution, which was "current" several steps before [11]. Here, each current solution still takes on the role of an acceptance benchmark, but it will be used at later steps. The net effect is that LAHC also considers poor solution as the basis for the next solution – an improvement of the general Hill Climbing algorithm as far as avoiding local minima/maxima problem.

The main component of our strategy LAHCS that constitutes the LAHC algorithm can be summarised in Figure 1.

Produce an initial solution s
Calculate initial cost function C(s)
for all *k ϵ {0...L-1} do* $\hat{C}k \leftarrow C(s)$
Assign the initial number of iteration I $\leftarrow 0$;
do until *a chosen stopping condition:*
  *Construct a candidate solution s**
  Calculate its cost function C(s)*
  $v \leftarrow I \bmod L$
  if $C(s^*) \leq \hat{C}v$
  then *accept candidate* ($s \leftarrow s^*$)
  Insert cost value into the list $\hat{C}v \leftarrow C(s)$
  Increment the number of iteration $I \leftarrow I+1$
end do
Sort (s)

Fig. 1 LAHCS Strategy

In order to solve the test redundancy problem with prioritisation, the following objective function has been considered.

$$min\ g = truncate\ f(x_1,x_2,....x_n) \quad (1)$$

where: x_1, x_2, ... x_n are different combinations of the solution sequence.

We need to find a truncated sequence of $(x_1,x_2,....x_n)$ that will give the optimal (minimal) value for the objective function $g(x)$ based on the order of the given weighted priority. Here, if each of the variable $(x_1,x_2,....x_n)$ can be chosen, this will yield n! = n*(n-1)*(n-2)…*(1) number of permutation sequences. Considering all exhaustive sequences, the searching process can take hours, days, or even weeks depending on the size of the problem.

A set of m random sequence is generated from $(x_1,x_2,....x_n)$ from $(n_1,n_2,....n_n)$ number of ways. The generated solution would be:

$$(x^k_1,x^k_2,....x^k_n) \text{ where } k = 1,2....m \text{ and } m \leq n \quad (2)$$

The fitness generated from $(x^k_1,x^k_2,....x^k_n)$ is then substituted in $g(x)$ to get the minimum cost function. Then, the most minimum solution is then sorted according to the weighted priority.

$$f(x^k_1) \leq f(x^k_2) \ ... \leq f(x^k_m) \quad (3)$$

In order to adopt LAHC as the basis algorithm for test redundancy reduction and prioritization, there is a need to choose the appropriate stopping condition as well the history length (L) that controls the memory of the previous cost functions. Here, the longer the history length, the longer the search and usually the better the results.

Theoretically, we argue that the stopping criteria should always be at least the same number of defined test case but must not be more than the factorial of the test suite size for reduction (i.e. n≤ stopping criteria ≤ n!). If the most minimum stopping criteria is less than test suite size, we cannot be sure that we have considered all the test cases in the test suite at least once for reduction. In similar manner, if the maximum stopping criteria is greater than the factorial of the test suite size, we might as well use exhaustive search.

The question now is that what is the best value for stopping criteria? Based on the aforementioned conditions consideration, we have decided to adopt the stopping criteria = n x L (where n = test suite size and L=the number of defined requirements) when (n! > n x L). In the case when (n! < n x L), then the best stopping criteria would be at n!

As for the history length (L), we argue that the value should be at least equal to the number of defined requirements. In this manner, we can be sure that priority ordering of requirement prioritization can still be possible should there be no reduction of test suite size. It should be noted that the aforementioned decisions on the stopping criteria and history length still adhere to the required condition, n≤ stopping criteria ≤ n!

IV. Benchmarking Experiments

To benchmark the performance of LAHCS against related work (including GE, GRE, and HGS), we have adopted the existing comparative case studies which are reported by Chen and Lau [3]. Additionally, we have also added 2 new case studies with sufficiently large test size and requirements. The detailed configurations are shown in Table I, Table II, Table III and Table IV respectively.

For the case studies 1 till 3, no priority is explicitly defined for GE, GRE, and HGS. For LAHCS, the priority is defined in order of requirements, that is, the lower order requirement has always higher priority than the subsequent requirement. As for the stopping criteria and history length, we use the sets of values according to our defined conditions given earlier. For Case Study 1, the value for stopping criteria = 19x7=133 and L = 19. For Case Study 2, the value for stopping criteria = 19x9=171 and L = 19. For Case Study 3, the value for stopping criteria = 19x12=228 and L = 19. For Case Study 4 and 5, the value for stopping criteria = 24x31=744 and L = 31. For both case studies, we compare LAHCS against our own implementation of GE derived from Chen and Lau [2]. Here, unlike earlier case studies where requirement priorities are in increasing order, different weighted requirements priorities are defined for LAHCS (i.e. the same priority for both case study 4 and 5 respectively).

TABLE I
BENCHMARK CASE STUDY 1

Req_i	T_n
req_1	$\{t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$
req_2	$\{t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$
req_3	$\{t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$
req_4	$\{t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$
req_5	$\{t_1,t_2,t_5,t_7\}$
req_6	$\{t_2,t_3,t_4,t_6\}$
req_7	$\{t_1,t_7\}$
req_8	$\{t_2,t_5\}$
req_9	$\{t_1,t_7\}$
req_{10}	$\{t_1,t_2,t_5,t_7\}$
req_{11}	$\{t_2,t_3\}$
req_{12}	$\{t_3,t_4,t_6\}$
req_{13}	$\{t_2,t_3\}$
req_{14}	$\{t_2,t_3\}$
req_{15}	$\{t_3,t_4,t_7\}$
req_{16}	$\{t_4,t_6\}$
req_{17}	$\{t_3,t_4\}$
req_{18}	$\{t_3,t_4\}$
req_{19}	$\{t_4,t_6\}$

TABLE II
BENCHMARK CASE STUDY 2

Req_i	T_n
req_1	$\{t_1,t_2,t_3,t_4,t_8,t_9\}$
req_2	$\{t_1,t_2,t_3,t_4,t_8,t_9\}$
req_3	$\{t_1,t_2,t_3,t_4,t_8,t_9\}$
req_4	$\{t_1,t_2,t_3,t_4,t_8,t_9\}$
req_5	$\{t_1,t_2,t_9\}$
req_6	$\{t_2,t_3,t_4,t_8,t_9\}$
req_7	$\{t_1\}$
req_8	$\{t_2,t_9\}$
req_9	$\{t_1\}$
req_{10}	$\{t_1,t_2,t_9\}$
req_{11}	$\{t_2,t_3,t_8\}$
req_{12}	$\{t_3,t_4,t_8,t_9\}$
req_{13}	$\{t_2,t_3,t_8\}$
req_{14}	$\{t_2,t_3,t_8\}$
req_{15}	$\{t_3,t_4,t_9\}$
req_{16}	$\{t_4,t_8\}$
req_{17}	$\{t_3,t_4,t_9\}$
req_{18}	$\{t_3,t_4,t_9\}$
req_{19}	$\{t_4,t_8\}$

Concerning collection of results, as LAHCS gives non-deterministic outputs, we repeat all our runs for all 5 case studies 20 times and choose the best results. The Table V depicts the results for the first three case studies whilst Table VI highlights the last two case studies involving the comparison between LAHCS against GE. Here, cells with the best results are shaded accordingly.

TABLE III
BENCHMARK CASE STUDY 3

Req_i	T_n
req_1	$\{t_1,t_3,t_4,t_5,t_6,t_8,t_{10},t_{11},t_{12}\}$
req_2	$\{t_1,t_3,t_4,t_5,t_6,t_8,t_{10},t_{11},t_{12}\}$
req_3	$\{t_1,t_3,t_4,t_5,t_6,t_8,t_{10},t_{11},t_{12}\}$
req_4	$\{t_1,t_3,t_4,t_5,t_6,t_8,t_{10},t_{11},t_{12}\}$
req_5	$\{t_1,t_5,t_{10},t_{11},t_{12}\}$
req_6	$\{t_3,t_4,t_6,t_8,t_{10},t_{12}\}$
req_7	$\{t_1,t_{10},t_{12}\}$
req_8	$\{t_5,t_{11}\}$
req_9	$\{t_1,t_{10},t_{12}\}$
req_{10}	$\{t_1,t_5,t_{10},t_{11},t_{12}\}$
req_{11}	$\{t_3,t_8,t_{10}\}$
req_{12}	$\{t_3,t_4,t_6,t_8,t_{12}\}$
req_{13}	$\{t_3,t_8,t_{10}\}$
req_{14}	$\{t_3,t_8,t_{10}\}$
req_{15}	$\{t_3,t_4,t_{12}\}$
req_{16}	$\{t_4,t_6,t_8\}$
req_{17}	$\{t_3,t_4,t_{12}\}$
req_{18}	$\{t_3,t_4,t_{12}\}$
req_{19}	$\{t_4,t_6,t_8\}$

TABLE IV
BENCHMARK CASE STUDY 4 AND 5

Priority	Req_i	T_n *for Case Study 4*	T_n *for Case Study 5*
0	req_1	$\{t_0,t_3,t_7,t_{18},t_{29}\}$	$\{t_0,t_3,t_7,t_{18},t_{19},t_{29}\}$
0	req_2	$\{t_3,t_{16},t_{22}\}$	$\{t_1,t_2,t_3,t_6,t_{12},t_{16},t_{22},t_{24}\}$
1	req_3	$\{t_0,t_2,t_{25},t_{27}\}$	$\{t_0,t_2,t_{25},t_{27}\}$
2	req_4	$\{t_{11},t_{30}\}$	$\{t_{11},t_{30}\}$
50	req_5	$\{t_1,t_4,t_8,t_{14},t_{25}\}$	$\{t_1,t_4,t_8,t_{14},t_{25}\}$
100	req_6	$\{t_9,t_{14},t_{19},t_{24}\}$	$\{t_9,t_{14},t_{19},t_{24}\}$
2	req_7	$\{t_5,t_{10},t_{21}\}$	$\{t_5,t_{10},t_{21}\}$
5	req_8	$\{t_4,t_{20}\}$	$\{t_4,t_{20}\}$
7	req_9	$\{t_7,t_{17},t_{24},t_{26}\}$	$\{t_7,t_{17},t_{24}\}$
8	req_{10}	$\{t_6,t_{15},t_{29}\}$	$\{t_{15},t_{29}\}$
90	req_{11}	$\{t_{10},t_{15},t_{23}\}$	$\{t_{10},t_{15},t_{23}\}$
80	req_{12}	$\{t_1,t_6\}$	$\{t_1,t_6\}$
45	req_{13}	$\{t_4\}$	$\{t_6\}$
67	req_{14}	$\{t_2,t_8,t_{13},t_{16},t_{23}\}$	$\{t_2,t_8,t_{13},t_{16},t_{23}\}$
55	req_{15}	$\{t_{28}\}$	$\{t_{20},t_{28}\}$
30	req_{16}	$\{t_{22},t_{28}\}$	$\{t_0,t_{18},t_{22}\}$
6	req_{17}	$\{t_{17},t_{29}\}$	$\{t_{17},t_{29}\}$
7	req_{18}	$\{t_5,t_{20}\}$	$\{t_5,t_{20}\}$
9	req_{19}	$\{t_9,t_{25}\}$	$\{t_9,t_{25}\}$
22	req_{20}	$\{t_{12}\}$	$\{t_{10},t_{12}\}$
12	req_{21}	$\{t_9,t_{28},t_{30}\}$	$\{t_9,t_{28},t_{30}\}$
46	req_{22}	$\{t_3,t_{24}\}$	$\{t_3,t_{24}\}$
76	req_{23}	$\{t_0,t_{30}\}$	$\{t_0,t_5,t_{30}\}$
19	req_{24}	$\{t_5,t_8,t_{11},t_{26},t_{27}\}$	$\{t_5,t_8,t_{11},t_{13},t_{26},t_{27}\}$

V. DISCUSSION

TABLE V
BENCHMARKING RESULTS FOR CASE STUDIES 1, 2, AND 3

Strategy	**Case Study 1**	**Case Study 2**	**Case Study 3**
GRE	$\{t_2,t_4,t_1(t_7)\}$ Reduction = 62.5%	$\{t_1,t_3,t_2(t_9), t_4(t_8)\}$ Reduction = 33%	$\{t_5(t_{11}),t_3,t_{10}(t_{12}),t_4(t_8)\}$ Reduction = 50%
GE	$\{t_3,t_1(t_7),t_4(t_6),t_2(t_5)\}$ Reduction = 50%	$\{t_1,t_3,t_2(t_9),t_4(t_8)\}$ Reduction = 33%	$\{t_{12},t_8,t_5(t_{11})\}$ Reduction = 66%
HGS	$\{t_3,t_1(t_7),t_4(t_6),t_2(t_5)\}$ Reduction = 50%	$\{t_1,t_4,t_2\}$ or $\{t_1,t_8,t_9\}$ Reduction = 50%	$\{t_5 (t_{11}),t_3,t_1(t_{10},t_{12}),t_4(t_6,t_8)\}$ Reduction = 50%
LAHCS	$\{t_1(t_7),t_2,t_4\}$ Reduction = 62.5%	$\{t_1,t_2,t_4\}$ or $\{t_1,t_8,t_9\}$ or $\{t_1,t_9,t_8\}$ Reduction = 50%	$\{t_5(t_{11}),t_8,t_{12}\}$ or $\{(t_5(t_{11}),t_{10},t_4\}$ Reduction = 66%

TABLE VI
BENCHMARKING RESULTS FOR CASE STUDIES 4 AND 5

Strategy	**Case Study 4**	**Case Study 5**
GE	*$\{t_4,t_{28},t_{12},t_5,t_3,t_2,t_6,t_9,t_{17},t_{10},t_{11}\}$* *Reduction = 64%*	*$\{t_6,t_0,t_5,t_9,t_4,t_{10},t_{17},t_2,t_3,t_{11},t_{15},t_{20}\}$* *Reduction = 61%*
LAHCS	*$\{t_7,t_{17},t_{12},t_3,t_{25},t_6,t_{30},t_{28},t_{15},t_4,t_5,t_{24},t_{23}\}$* *Reduction = 58%*	*$\{t_7,t_{29},t_{11},t_3,t_{16},t_{20},t_0,t_{10},t_9,t_6,t_8\}$ or $\{t_{29},t_{27},t_{18},t_{28},t_{20},t_{30},t_{10},t_9,t_6,t_8,t_{24}\}$* *Reduction = 64%*

Referring to the results in Table V and VI, a number of observations can be elaborated further. The first observation relates to the adoption of Hill Climbing as the main basis for LAHCS. While Hill Climbing algorithm has always been criticized for its proneness to get trap into local minima/maxima, the development of LAHCS has proven that Late Acceptance feature within Hill Climbing significantly improves its performance owing to the balance selection between intensification (i.e. how intensive is the local search for the current solution is) and diversification (i.e. how diverse is the current solution). Here, all solutions whether good or inferior solution are also considered for accepting new neighbourhood solution – unlike Simulated Annealing which adopts probabilistic criteria based on Boltzmann energy function [12].

From all the case studies, LAHCS produces sufficiently competitive results in terms of percentage of reduction (see all the shaded cells) although in different order owing to its weighted prioritization order. With the exception of Case Study 4, LAHCS is able to match the best performing strategies (as in Case Study 1 and 2) and even outperforms its competitors (as in Case Studies 3 and 5 respectively). Specifically, for Case Study 3 and 5, LAHCS is also able to produce diversified solutions not found by other strategies. Also, for Case Study 5, the percentage of reduction for LAHCS outperforms that of GE but, in return, GE outperforms LAHCS for Case Study 4 suggesting that there is no single one size fit all strategy for test redundancy reduction.

Another observation relates to prioritization. The question is how prioritization can be effectively captured in order to order the suite accordingly. In general, any requirement prioritization can be defined in term of Likert scale. In this case, requirement priority can come directly from the stakeholder's (i.e. through specification documents) or from pragmatic experiences of the engineers on the likely hood of failure of each requirement and its impact (i.e. through (normalized) priority = likelihood x impact) [13]. In many cases, software testing activities get squeezed towards the end resulting from (unplanned) extension of other software development activities. Owing to the need to accommodate market demands and constraints, test engineers are often required to prioritize only critical test cases that have the highest impact for testing consideration.

Finally, test reduction strategy serves two sides of the same coin. On one side of the coin, the strategy involved must be able to generate the most optimal and minimum number of test cases in order to reduce testing costs. On the other side of the coin, the strategy must also not sacrifice the bug-detection capabilities using lesser number of test cases. When dealing with any testing strategy, test engineers may be poised with crossroad decisions, that is, to minimize as much as possible or to keep some if not all test cases. In some cases, it is important to test all highly critical requirements multiple times (i.e. voluntary redundancies) with more than one test case, that is, to ensure strict adherence to specification. In such a case, test engineers are free to include such test cases as required in the final test suite list (i.e. seeding). In similar manner, test engineers are also free to forbid a set of test cases if such a need arises (i.e. constraints). To make matters worse, there a no hard rules as all decisions depend on circumstances as well as the creativity and judgment of test engineers based on the testing job at hand as well as the testing in context.

VI. CONCLUSIONS

Summing up, this paper has elaborated a new strategy, called LAHCS, based on Late Acceptance Hill Climbing Algorithm. Our experience with LAHCS has been promising. As the scope for future work, we are looking into improving LAHCS to address reduction with multi-objective consideration along with the support voluntary redundancies, constraints and seeding.

ACKNOWLEDGMENT

This research work involves collaborative efforts between Universiti Malaysia Pahang and Umm Al-Qura University. The work is funded by grant number 11-INF1674-10 from the Long-Term National Plan for Science, Technology and Innovation (LT-NPSTI), the King Abdul-Aziz City for Science and Technology (KACST), Kingdom of Saudi Arabia. We thank the Innovation Office, UMP and the Science and Technology Unit at Umm Al-Qura University for their continued logistics support.

REFERENCES

[1] M. J. Harrold, N. Gupta, and M. L. Soffa, "A Methodology for Controlling the Size of a Test Suite," *ACM Transactions on Software Engineering Methodology*, vol. 2, pp. 270-285, 1993.

[2] T. Y. Chen and M. F. Lau, "Heuristics towards the Optimization of the Size of a Test Suite," in *3rd International Conference of Software Quality Management*, Seville, 1995, pp. 415-424.

[3] T. Y. Chen and M. F. Lau, "A New Heuristic for Test Suite Reduction," *Information and Software Technology*, vol. 40, pp. 347-354, 1998.

[4] V. Chvatal, "A Greedy Heuristic for Set Covering Problem," *Mathematics of Operations Research*, vol. 4, pp. 233-235, 1979.

[5] X. Shengwei, M. Huaikou, and G. Honghao, "Test Suite Reduction Using Weighted Set Covering Techniques," in *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing*, Kyoto, 2012, pp. 307-312.

[6] B. Galeebathullah and C. P. Indumathi, "A Novel Approach for Controlling a Size of a Test Suite with Simple Technique," *International Journal of Computer Science and Engineering*, vol. 2, pp. 614-618, 2010.

[7] S. Tallam and N.Gupta, "A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization," presented at the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Lisbon, Portugal, 2005.

[8] P. Ng, R. Y. K. Fung, and R. W. M. Kong, "Incremental Model-Based Test Suite Reduction with Formal Concept Analysis," *Journal of Information Processing Systems*, vol. 6, pp. 197-208, 2010.

[9] E. K. Burke and Y. Bykov, "The Late Acceptance Hill-Climbing Heuristic," Technical Report CSM-192, Computing Science and Mathematics, University of Stirling 2012.

[10] E. K. Burke and Y. Bykov, "A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems," in *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 08)*, Montreal, Canada, 2008.

[11] Y. Bykov. (2014, May 7). *Late Acceptance Hill Climbing Algorithm*. Available: http://www.yuribykov.com/LAHC/

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

[13] R. Black, *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*: Wiley, 1999.