Concurrently Searching Branches in Software Tests Generation through Multitask Evolution

Ramon Sagarna
Rolls-Royce@NTU Corporate Lab c/o
School of Computer Science and Engineering
Nanyang Technological University
Singapore 639798
Email: saramon@ntu.edu.sg

Ong Yew-Soon
Rolls-Royce@NTU Corporate Lab c/o
School of Computer Science and Engineering
Nanyang Technological University
Singapore 639798
Email: asysong@ntu.edu.sg

***Abstract*—Multitask evolutionary computation (MT-EC) has been recently identified as a potentially useful paradigm for significant real-world domains. One such domain is the field of software testing. Although a number of evolutionary approaches exist already, there is a lack of strategies that can leverage knowledge from different sources to enhance the search process. In this work, we focus on branch testing and explore the capability of MT-EC to guide the search by exploiting inter-branch information. To the best of our knowledge, this is the first application of MT-EC to real-world problems with more than two tasks. Precisely, we evince that selection, together with the preference relation used to compare individuals, form a mechanism capable of achieving a concurrent search for the branches while exploiting inter-branch knowledge in the process. Further, we demonstrate that the intensity of the transfer can be altered with the implemented selection. The experimental results on benchmark programs suggest that MT-EC can be specially useful in situations where the budget for the search process is limited.**

1. Introduction

Recently, the potential of multitask optimization to address application domains of practical significance has been put forward [1]. Different to multiobjective optimization, in the multitask scenario we are given a set of functions and the goal is to find a set of solutions, each of which optimizes a function.

A prominent real-world problem that remarkably fits in with this scenario arises from the domain of software testing. Namely, the generation of the inputs to be applied to a program under test is a main concern. Since exhaustive testing is cost prohibitive in general, test inputs are sought with the aim of satisfying an adequacy criterion that addresses a particular aspect of the program [2], [3]. This often turns the generation of the inputs into a non-trivial task. Further, if performed manually - as it remains largely in the industry - it becomes time-consuming, labor-intensive and error-prone [3], [4], [5], so efforts have been addressed towards its automation. One alternative that has proven effective over the last decade and a half is Search Based Software Test Data Generation (SBSTDG) [5], which relies on the usage of search heuristics for the selection of the appropriate test inputs [6], [7], [8]. Whilst a number of testing criteria have been faced, branch testing has been the main subject of study to date [9], [10]. This criterion is broadly used, it is part of well-known standards such as ANSI/IEEE 1008-2002 [11] and RTCA/DO-178B [12], and its relevance prevails nowadays [13].

In spite of their success, SBSTDG generators are usually advanced randomized heuristics for which a complete analysis is non-trivial. Since this hampers the design of improved approaches, the SBSTDG community has called for feasible strategies to select or adapt the search method [10], [14]. Casting the software testing problem as multitask optimization, one direction to study is the exploitation of the inter-tasks parallelism by leveraging the information of a task that is potentially beneficial for another. Interestingly, we observe that one of the earliest and most popular SBSTDG approaches already makes use of such inter-task parallelism during the search [7]. However, we find that inter-task information is not used intensively to guide the search of the tasks in a concurrent manner. This makes the method rather restrictive in the way of exploiting the relationships among tasks. Further, the information exchange across tasks is handled through archives, which adds to the complexity of the design and of the parameter tuning.

Recently, a number of research efforts based on evolutionary computation have shown highly effective in leveraging inter-task knowledge during the search [1]. Such multitask evolutionary computation (MT-EC) approaches depict a flexible framework relying on evolutionary mechanisms for concurrently evolving and actively transferring the information across tasks. In the present work, we propose a MT-EC approach to SBSTDG as a means to achieve a simplified multitask generator. We will restrict ourselves to branch testing, although the approach can be extended to other testing criteria. To the best of our knowledge, this is the first attempt to apply MT-EC to a real-world problem with more than two tasks. It thus contributes to demonstrating the feasibility of this paradigm for real-world scenarios.

Aiming at simplified multitasking, we also contribute to elucidating the evolutionary mechanisms by which effective inter-task transfer can be achieved within the MT-EC framework. In previous works, such transfer has been shown to be attained through variation operators, e.g. crossover [1]. Here, we unveil selection as another alternative. More precisely, we evince that the preference relation used to compare individuals, jointly with the selection operator, but with no additional evolutionary operators, allow to accomplish inter-task transfer in an effective search. Further, we demonstrate that the intensity of the transfer can be altered with the implemented selection. In particular, we evaluate two extreme selection schemes: one based on the evaluation of one task only, and another based on all the tasks, and observe that the former leads to lower transfer intensity than the latter.

The rest of the paper is organized as follows. Section 2 introduces the SBSTDG approach to branch testing, together with the early multitask approach in [7]. In Section 3.3, after motivating MT-EC for SBSTDG, some basic concepts on MT-EC are outlined, followed by a description of the two selection schemes and the MT-EC algorithm we employ. Section 4 presents an empirical study on the effectiveness of MT-EC for branch testing, and on the impact of the selection on inter-task transfer. We finish with concluding remarks in Section 5.

2. Search-based Software Test Data Generation

The field of Search Based Software Test Data Generation (SBSTDG) aims at selecting the test inputs that fulfill a testing criterion by making use of search heuristics in the process [5]. Hereafter, we concentrate on branch testing.

Given the code of a program, a *branch* refers to one of the truth values of the expression in a conditional statement [15]. The *coverage* of a branch alludes then to a program execution with an input such that the control flow brings about the branch. In the branch testing problem, the goal is to find a subset of the input domain maximizing the number of branches covered. Note that the set of solutions for this optimization problem may be huge, but essentially finite due to computational limits for data representations. Typically, the value of a solution is given by the percentage of covered branches. This does not imply though that 100% coverage is optimal, as a branch may be *infeasible*, i.e. no input can cover it. Since the problem of discovering infeasibility is undecidable [2], [15], the true optimal percentage is unknown in general and 100% becomes just an upper bound.

It is sufficient for a solution of the branch testing problem to have one test input per branch. Thus, the idea underlying many SBSTDG works is to seek the coverage of each branch by assigning it an optimization problem. We follow here a widespread formulation (see [5], [9] and references therein).

2.1. Branch Coverage Optimization Problem Formulation

In order to work with a comprehensive model, we will represent the potential variations of the control flow with a *control flow graph* (CFG) (V, A), where V is the set of vertices and $A \subseteq V \times V$ is the set of arcs [16]. Each vertex in V is a code basic block, excepting two vertices labeled s and e, which refer to the program entry and exit respectively. A conditional statement maps into a vertex with $outdegree$ bigger than one, which we call *decision vertex*. A branch is represented by an arc incident from a decision vertex. Likewise, an *execution path* represents the actual control flow of the program when executed with a given input.

Given the target branch t and the input domain of the program, Ω, the goal is to find $x^* = \arg\min_{x \in \Omega} f(x)$, with $f : \Omega \longrightarrow \mathbb{R}_{\geq 0}$ defined so as to provide a score of how close the execution path of an input is to a path reaching t. This assessment relies on the *constraint branches* that are deemed to be covered in a path to t. These are not all the branches, but those whose decision vertex is related to t through a control dependence relation. A number of meanings have been employed in the literature for this relation [5]. We will resort here to a notion unveiling branches that, if taken, t is missed for sure; hence a sibling branch should be followed.

Definition 1. Given a CFG $G = (V, A)$, let $v, w \in V$. v is *critical* for w if there are two arcs $(v, v_1), (v, v_2)$, $v_1 \neq v_2$, such that in every path p_1 from v_1 to e, $w \notin p_1$, and a path p_2 from v_2 to w exists.

The following remarks facilitate the description of the objective function f.

Remark 1. Given a target $t = (o, o')$, in any execution path p such that $t \notin p$, there is a critical vertex for o inducing an uncovered constraint branch.

Remark 2. The uncovered constraint branch referred to in Remark 1 is the only one iff for every critical vertex v in p, $outdegree(v) = 2$. That is, if $outdegree > 2$ for some critical vertex then several uncovered constrained branches exist. Figure 1 illustrates such a scenario.

f is obtained by combining the so-called *approach level* and *branch distance* measures [7].

Given a target $t = (o, o')$, an input $\boldsymbol{x}$ and its execution path $p_{\boldsymbol{x}}$, the approach level, $l_o(\boldsymbol{x})$, returns the minimum number of uncovered constraint branches lying in a path from $p_{\boldsymbol{x}}$ to t. This is achieved by looking at each critical vertex $c \in p_{\boldsymbol{x}}$ that induces an uncovered constraint branch, and counting the uncovered constraint branches lying between c and o.

The branch distance, $d_c(\boldsymbol{x})$, points at the expression in the conditional statement of c, and measures how far it was from evaluating to the branch that is heading for t. For example, let $A > B$ be the expression, with A and B defined as numerical variables. If $A_{\boldsymbol{x}}$ and $B_{\boldsymbol{x}}$ denote their respective values in the execution of the program with $\boldsymbol{x}$, then $d_c(\boldsymbol{x}) = |A_{\boldsymbol{x}} - B_{\boldsymbol{x}}| + 1$ is typically chosen. In the case of having a compound expression, or the variables taking

other data types, alternative distances have been proposed (e.g. see [17]).

The objective function aggregates the two measures above as follows:

$$f(\boldsymbol{x}) = \begin{cases} l_o(\boldsymbol{x}) + \frac{d_c(\boldsymbol{x})}{M} & \text{if } t \text{ not covered} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where M is a normalization term which serves to guarantee that an input with a lower approach level than another receives a smaller function value.

Note that, in order to assess $f(\boldsymbol{x})$, we need to know the execution path of $\boldsymbol{x}$ (for $l_o(\boldsymbol{x})$) and the data produced during program execution (for $d_c(\boldsymbol{x})$). This information is acquired by assembling an instrumented version of the program, and running it with the input. This implies that, in general, f is not available in closed form. Hence, it is common to resort to a black-box search method to address the problem. In particular, the genetic algorithm has been the method of choice so far [5], [9].

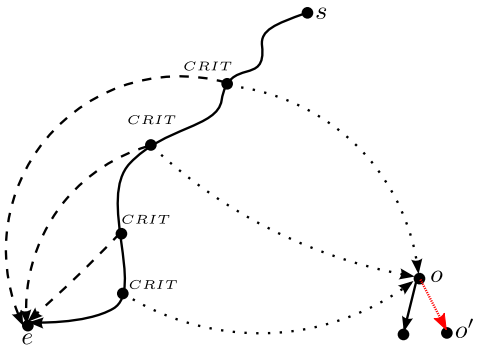

Figure 1. Illustration of an execution path with several critical vertices. Dotted lines stand for a path to the target, and dashed lines indicate all paths from the corresponding vertex miss o.

2.2. A 2-Step Approach

The early approach by Wegener et al. [7] constitutes one of the most popular SBSTDG works to date. This approach consists of a two-step iterative process in which, at each round, firstly, a branch is chosen and marked as a target, and secondly, a search algorithm is run to solve the optimization problem of this target. This implies the search algorithm deals with one optimization problem at a time, however the real goal is to solve a set of problems. The approach takes account of this by not only evaluating an input for the current target branch, but also with regard to all the others. For each branch, a set storing the best inputs so far is kept. In the target branch selection step, the branch with a highest quality set of inputs is chosen, and in the optimization step, this set is employed to initialize the search algorithm. By doing so, at each round, we address the optimization problem with the most promising initial solutions available. Both target selection and optimization steps are repeated until 100% branch coverage is attained or all branches have been treated.

3. An Evolutionary Multitasking Approach to SBSTDG

The motivation for a multitasking approach to branch testing can be realized by observing that the objective function in Equation 1 is total. This can be easily noticed by revisiting Remark 1 above which states that an uncovered critical branch always exists. Since all the objective functions share the same search space (the input domain), every input can be evaluated for any target branch. An execution path closer to some targets than to others should enclose some form of information that makes it superior for the former in terms of uncovered constrained branches. It makes sense then to try to leverage such information through a multitask search.

The 2-step SBSTDG approach described in Section 2.2 can be seen as a multitask greedy algorithm where, at each step, the addition of a new input to the final solution is checked. This input is generated during the search for the coverage of the target branch, yet it may solve other branches or be suitable for future searches. The SBSTDG generator can be viewed then as a 2-level search: in the lower level, the target branch is searched for making use of a black-box algorithm, whereas in the upper level, inputs sampled in the lower level are handled with a view to solving the whole problem. Apart from adding to the complexity in terms of algorithmic design and parameter tuning, we find that such a 2-level search makes a restricted use of the relationships among tasks. These relationships are not employed to guide the search at the lower level, i.e. there is no competition among tasks during the search. It is only at the upper level, when the next target branch has to be chosen, that tasks compete for the next search effort.

We propose here a MT-EC approach as a means to achieve a concurrent search for the tasks while exploiting inter-task knowledge in the process. An MT-EC algorithm keeps a population of individuals, each of which is evaluated considering a number of tasks. Since the fitness of two individuals may be given by different tasks, an active competition among the latter takes place during selection. This way, search efforts are dynamically allotted towards the most promising tasks at each moment in time.

3.1. MT-EC Concepts

In [1], Gupta et al. presented the *multifactorial evolutionary algorithm*, a general MT-EC method for multitask optimization. As part of the approach, the authors also introduced a number of basic concepts to formally handle solutions in a MT-EC framework. In particular, the preference relation upon which individuals are compared is of relevance to us. The next definitions are adapted from their work. All of them consider a problem with k functions to be minimized, $T = \{T_1, ..., T_k\}$, and a population P of individuals.

Definition 2. The *factorial rank* r_j^i of $I_i \in P$ for a task $T_j \in T$ is the index of I_i in the list of individuals from

P sorted in ascending order with respect to their value in T_j.

Definition 3. The *skill factor* τ_i of $I_i \in P$ is the one task from T assigned to I_i.

Definition 4. The *scalar fitness* ϕ_i of $I_i \in P$ is $\phi_i = 1/r^i_{\tau_i}$.

Two individuals I_1 and I_2 are compared through their respective scalar fitness: $I_1 \prec I_2 \; iff \; \phi_1 > \phi_2$.

Building upon the definitions above, a preference relation can be simply specified by taking $\tau_i = argmin_{j \in S}\{r^i_j\}$, where $S \subseteq T$ is the subset of tasks being considered for the individual I_i. If S is the outcome of a mapping $\delta : P \longrightarrow 2^T$, the set of possible δ describes a family of preference relations that compare individuals on the basis of the task they best perform in (τ_i).

3.2. Task Selection Schemes

The selection operator that has been implemented in previous works consists of a mapping δ returning exactly one task [1]. The rationale behind this choice is to keep the computational cost of the multitask approach within practical bounds. We observe however that in some real-world problems, like the present one, the number of tasks does not place a significant burden. Particularly, in SBSTDG, the main cost of the evaluation comes from the execution of the program, which can be performed once for all the tasks (branches). This opens the range of feasible definitions for the δ mapping above, which in turn leads to different preference relations. Aiming at studying the influence of the relations on inter-task transfer, we will restrict ourselves to two extreme cases:

mtec-all All the unsolved tasks are considered for every individual.

mtec-one Just one task is considered; the skill factor of the parent.

An important add-on of both schemes is that they incorporate a *task-reassignment* mechanism. That is, whenever a task has been solved, it is dropped from the search, the optimum is stored, and in the case of mtec-one, the individuals whose skill factor was the solved task are reassigned a random task and re-evaluated. As it could be noted, in SBSTDG, this re-evaluation comes at no significant extra cost, since we can compute the objective function value upon the information collected from the program execution without the need to re-run.

We adduce that, in real-world applications, two situations may arise in which a task-reassignment is desirable:

- There is a potential risk of the population to converge towards solved tasks.
- Starting from a uniformly random initial population, it may well happen that, because of the inter-task competition during selection, no task survives as the skill factor of an individual. In the case of mtec-one, this implies the search for those tasks is abandoned.

3.3. A MT-EC Algorithm for SBSTDG

We intend to evaluate the sufficiency of the selection schemes described above. Therefore, we completely disregard crossover in our MT-EC approach. The only variation comes from the mutation operator. The pseudo-code in Algorithm 1 describes the implementation we have adopted.

Algorithm 1 Pseudo-code for the MT-EC approach.

```
1:  P_0 ← Generate initial population of N random individuals
2:  Evaluate each individual in P_0 for all tasks
3:  For each I_i ∈ P_0 compute skill-factor τ_i and scalar fitness φ_i considering unsolved tasks only
4:  for t=0 ... until stopping criteria met do
5:      P_t^c ← Mutate each I_i ∈ P_t
6:      Evaluate each individual in P_t^c according to the
7:      selection scheme
8:      if New tasks solved then
9:          Mark tasks as solved
10:         if Selection scheme is one-task then
11:             Reassign tasks to individuals in P_t and P_t^c,
12:             and re-evaluate
13:         end if
14:     end if
15:     For each I_i ∈ P_t ∪ P_t^c compute skill-factor τ_i and
16:     scalar fitness φ_i considering unsolved tasks only
17:     P_{t+1} ← Select the N fittest individuals from P_t ∪ P_t^c
18: end for
```

The initial population consists of individuals generated uniformly at random which undergo the mtec-all scheme for their evaluation. Once evolution commences, the chosen selection scheme is followed (lines 6-17).

In order to rationalize the achievement of an inter-task transfer of information through this algorithm, we need a description in precise terms. Given one parent individual I_p and its child I_c, we say that *inter-task transfer from I_p to I_c* has taken place if $\tau_p \neq \tau_c$. Likewise, *generational inter-task transfer from I_p to I_c* has occurred if $\tau_p \neq \tau_c$ and I_c survives to the next generation or solves a task. The former notion designates the event in which some information has been exchanged across tasks, while the latter alludes to a transfer that can be potentially useful in the search for the recipient task(s). Grounded on these descriptions, we can realize that, in the mtec-one scheme, transfer will be triggered just when a task is solved, owing to the task-reassignment mechanism. By contrast, it seems intuitive that the mtec-all selection favors a more intense exploitation of inter-task parallelism and facilitates generational transfer. Accordingly, it could be argued that these two schemes promote opposite degrees of inter-task transfer during the search. We will conduct next an empirical study to find further support for our rationale.

4. Experimental Evaluation

The following experimental study aims at two main goals: (i) to evaluate the effectiveness of a MT-EC approach

for branch testing, and (ii) to assess the sufficiency of the selection strategy as a mechanism to achieve inter-task transfer during the search.

4.1. Experimental Setup

We consider a benchmark of 10 numerical calculus functions written in C, extracted from the book *Numerical Recipes in C. The Art of Scientific Computing* [18]. The number of branches in these functions ranges from 16 to 44. The input parameters are of type integer or real, each function taking between 3 and up to 67 parameters. We represent an input with a bit string of length n given by the number of parameters.

To evaluate the feasibility of the MT-EC approach as an alternative for software testing we compare it against the 2-level SBSTDG method from Section 2.2 and an independent optimization for each of the branches. It is important to note that all these method share the same initialization, i.e. only the branches that remain uncovered after this step are sought. The same mutation-based evolutionary algorithm is employed by all the methods. In particular, the mutation consists of a standard bit flip with probability $1/n$; mutation of at least one gene is enforced. The population size is set to $2n$. In the 2-level method, the size of the auxiliary pool of individuals is the same as the population size. As for the stopping criterion, each method halts if all the branches have been covered, 30000 inputs are evaluated (program executions), or a maximum number of 100 generations per uncovered branch is reached. All the results reported here are the average over 20 runs of the corresponding method and program.

4.2. MT-EC versus classical SBSTDG

As pointed above, during the initialization of the population a number of branches will necessarily be covered. It is the set of remaining branches that forms the multitask optimization problem. The second and third columns in Table 1 show the total number of branches of each program and the actual average number of branches being sought, respectively. It can be observed that in many cases the amount of tasks being faced is well above two, indicating the larger scale of the multitask application.

Table 1 also presents the coverage percentage achieved by the four methods in the comparison. A first observation is that, in almost all the programs, the three approaches implementing a multitask strategy show superior or equal performance than the method based on independent searches. This serves to highlight the potential of exploiting inter-task information during the search for the appropriate inputs.

Further looking at the table, the MT-EC approaches seem slightly superior, although no apparent difference can be appreciated. We recall that these are final results after the stopping criteria described above have been met. A closer look is displayed through the convergence plots shown in Figures 2-5. The trends shown herein are similar to those in the rest of the programs. In particular, Figure 2 is representative of the behavior observed in *fit*, *laguer* and *bnldev*. It is clear from Table 1 that in the first two, MT-EC outperforms the 2-level method. The figure reveals that for *bnldev* MT-EC converges substantially faster than 2-level.

Figure 3 depicts the behavior noted in *adi*, *gaussj*, *toeplz* and *plgndr*. No clear difference is observed in the convergence of the multitask methods, however the independent searches approach lags visibly behind, until eventually moves closer to 100% coverage.

In Figure 4, a scenario of no apparent dissimilarity among any of the methods is shown. However, this time all the contenders achieve full coverage and, as can be observed, the multitask methods converge faster than the other.

Finally, Figure 5 brings out the negative behavior of mtec-all in program *des*. As previously, the multitask approaches present a fast initial convergence, although in this case, mtec-all stagnates. Finding the cause for such performance is not easy. We hypothesize that, assuming mtec-all undergoes a more active inter-task transfer than the other three methods, in this problem instance a more focused search on the task may be more adequate.

Wrapping up, the outcomes from these experiments seem to indicate that MT-EC tends to quickly achieve upper levels of coverage, although it can later be caught up by other approaches. This would point at MT-EC as a promising alternative in situations where the budget that can be allotted to the search process is limited.

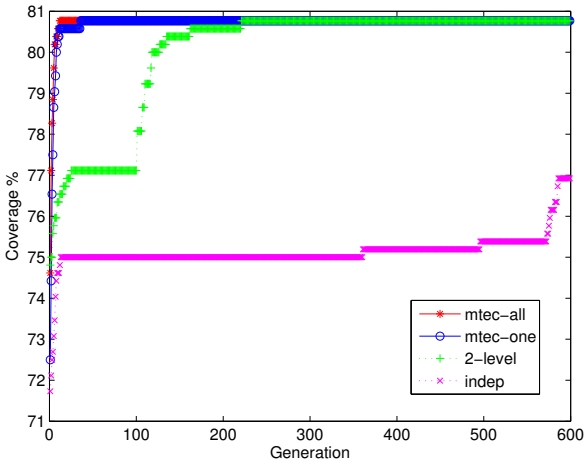

Figure 2. Evolution of coverage for program *bnldev*.

4.3. Selection for Inter-Task Transfer

In this section, we follow the notion of inter-task transfer and generational inter-task transfer specified in Section 3.3 in order to quantify the amount of information that has been exploited during the search by the MT-EC method.

TABLE 1. RESULTS OF EXPERIMENTS COMPARING MT-EC WITH STANDARD SBSTDG. SECOND AND THIRD COLUMNS SHOW THE TOTAL NUMBER OF BRANCHES AND THE NUMBER OF TASKS. THE NEXT COLUMNS PRESENT THE COVERAGE PERCENTAGE ACHIEVED BY EACH METHOD; BEST VALUES ARE MARKED IN BOLD.

Program	Branches	Tasks	indep	2-level	mtec-one	mtec-all
plgndr	20	3.25	99.583335	**100**	**100**	**100**
gaussj	42	1.6	**97.619**	**97.619**	**97.619**	**97.619**
toeplz	20	3.75	84.75	**85**	**85**	**85**
bessj	18	3.1	**100**	**100**	**100**	**100**
bnldev	26	7.45	76.923055	**80.7692**	**80.7692**	**80.7692**
des	16	2.8	93.4375	**93.75**	93.4375	91.875
fit	18	8.1	92.777775	95	**97.5**	**97.5**
laguer	16	4.7	85	90.3125	**91.25**	90.9375
sparse	30	5.55	88.000005	89.833335	81.333305	**90**
adi	44	19.25	56.249995	**59.0909**	**59.0909**	**59.0909**

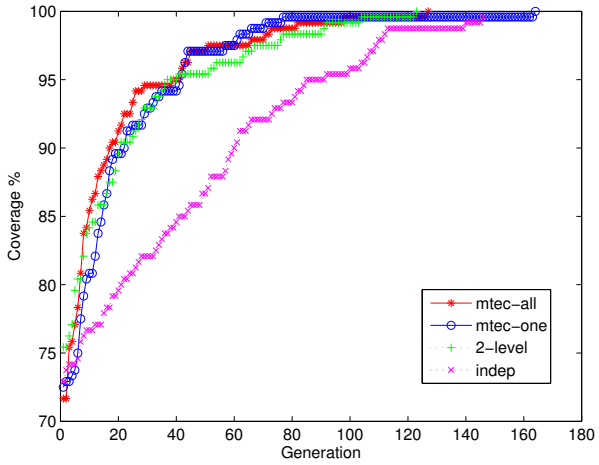

Figure 3. Evolution of coverage for program *plgndr*.

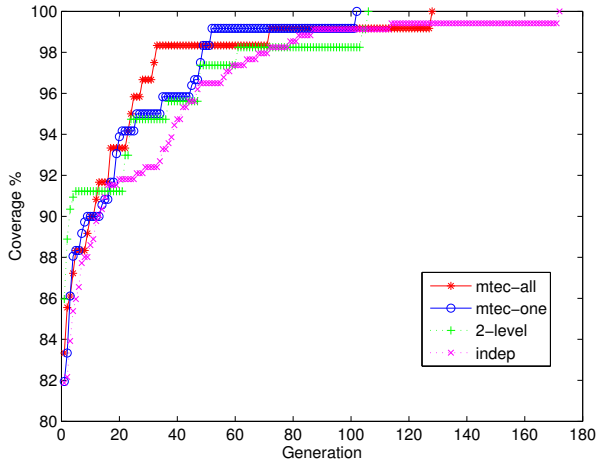

Figure 4. Evolution of coverage for program *bessj*.

Table 2 shows these quantities for both mtec-one and mtec-all on our benchmark programs. As can be observed, in almost all the cases, mtec-all produces a fairly larger amount of transfers and generational transfers than mtec-one. Interestingly, there exist two programs in which this does not hold. From an examination of the search trace, we see that, in *bessj*, the population of mtec-all forms niches that evolve rather in isolation. However, in mtec-one, whenever a task has been solved an amount of transfer equivalent to the number of individuals with that skill factor is enforced. As for program *fit*, mtec-all almost immediately achieves its peak coverage and covers all the branches but one, which disables any transfer.

While some exception seems to exist, in general terms, these outcomes support the intuition that the mtec-all scheme tends to favor a larger transfer intensity during the search than mtec-one. Figure 6 provides further support to this intuition by showing the number of generational transfers across the generations for program *des*. mtec-one only produces new transfers at the generations where a new branch is covered (compare with Figure 5). By contrast, mtec-all keeps leveraging information from other tasks that is subsequently transferred to the next population, even once it has converged. In this instance, however, such continuous generational transfer does not seem to help boost the coverage.

Nonetheless, in light of the performance results, these experiments serve to demonstrate that the selection mechanism alone can succeed in achieving effective inter-task transfer.

TABLE 2. NUMBER OF INTER-TASK TRANSFERS AND GENERATIONAL TRANSFERS PERFORMED FOR EACH PROGRAM.

	mtec-one		mtec-all	
Program	Transfer	GenTransfer	Transfer	GenTransfer
plgndr	101.45	68.8	114.75	61.75
gaussj	108	77.2	173.65	150.05
toeplz	15	4.85	6089.4	405.7
bessj	60.45	22.5	18	9.25
bnldev	39.45	15.95	12339.7	2739.55
des	172	106.2	1708.05	1254.8
fit	354.9	93.1	0	0
laguer	233.8	51	3269.8	257.5
sparse	0	0	12204.9	8108.55
adi	20.35	17.6	3555.8	2777.65

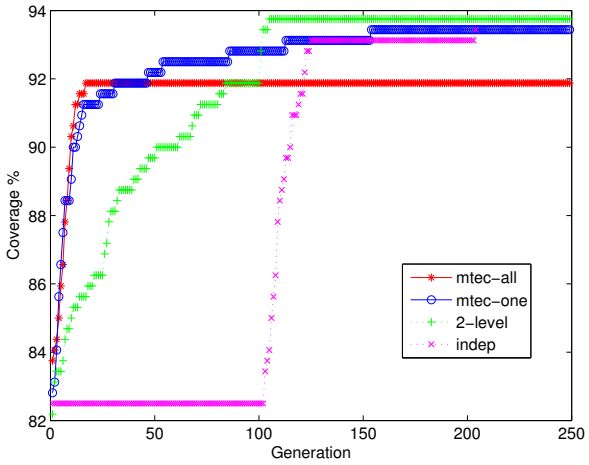

Figure 5. Evolution of coverage for program *des*.

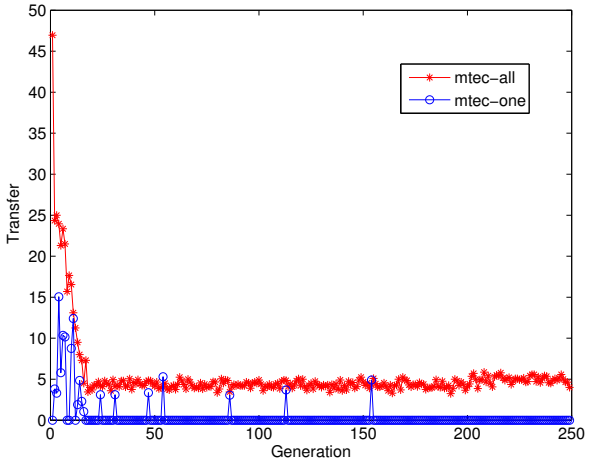

Figure 6. Amount of generational transfer over generations for program *des*.

5. Conclusion

In this work, we have proposed a multitask evolutionary computation approach to software testing. To our knowledge, this has been the first effort demonstrating the feasibility of this paradigm for solving real-world problems with more than two tasks. Precisely, we have advocated here for the selection strategy as a sufficient evolutionary mechanism to actively exploit the inter-relations of the tasks (branches) during the search. The outcomes from our experimental investigation support this capability. Further, the results suggest that the multitask approach may be specially well suited in testing scenarios in which a limited amount of resources can be allotted to the search algorithm. Given these encouraging results, a possible avenue for future research could be to leverage information from the problem domain in order to select the most promising tasks on which to focus on.

Acknowledgments

This work was supported by the Rolls-Royce@NTU Corporate Laboratory from the National Research Foundation, Singapore, under the Corp Lab@University Scheme.

References

[1] A. Gupta, Y. S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2016.

[2] B. Beizer, *Software Testing Techniques*. New York: Van Nostrand Rheinhold, 1990.

[3] A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering*. London: Pearson Education Limited, 2003.

[4] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Proceedings of the International Conference on Software Engineering, Workshop on the Future of Software Engineering*, L. C. Briand and A. L. Wolf, Eds. Washington, DC, USA: IEEE CS Press, 2007, pp. 85–103.

[5] P. McMinn, "Search-based software test data generation: a survey," *Software Testing Verification and Reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[6] E. Alba and F. Chicano, "Observations in using parallel and sequential evolutionary algorithms for automatic software testing," *Computers & Operations Research*, vol. 35, pp. 3161–3183, 2008.

[7] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, no. 14, pp. 841–854, 2001.

[8] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener, "Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 453–477, 2012.

[9] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," King's College London, Department of Computer Science, Tech. Rep. TR-09-03, 2009.

[10] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1–12.

[11] ANSI/IEEE 1008-2002, *IEEE Standard for Software Unit Testing: An American National Standard*. IEEE Standards Board, American National Standards Institute, 2002.

[12] RTCA/DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*. Radio Technical Commission for Aeronautics, 1992.

[13] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. Alipour, and D. Marinov, "Guidelines for coverage-based comparisons of non-adequate test suites," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 4, 2015.

[14] R. Sagarna, A. Mendiburu, I. Inza, and J. Lozano, "Assisting in search heuristics selection through multidimensional supervised classification: A case study on software testing," *Information Sciences*, vol. 258, pp. 122–139, 2014.

[15] P. Frankl and E. J. Weyuker, "A formal analysis of the fault-detecting ability of testing methods," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 202–213, 1993.

[16] N. E. Fenton, "The structural complexity of flowgraphs," in *Graph Theory with Applications to Algorithms and Computer Science*, Y. Alavy, G. Chartrand, L. Lesniak, D. R. Lick, and C. E. Wall, Eds. New York: John Wiley & Sons, 1985, pp. 273–282.

[17] N. Tracey, J. Clark, K. Mander, and J. McDermid, "An automated framework for structural test-data generation," in *Proceedings of the 13th IEEE Conference on Automated Software Engineering*, D. Redmiles and B. Nuseibeh, Eds. IEEE CS Press, 1998, pp. 285–288.

[18] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C. The Art of Scientific Computing*. New York: Cambridge University Press, 1988.