Front. Comput. Sci. China 2009, 3(1): 64–72
DOI 10.1007/s11704-009-0006-6

REVIEW ARTICLE

# Runtime analysis of search heuristics on software engineering problems

**Per Kristian LEHRE(✉), Xin YAO**

The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Birmingham B15 2TT, United Kingdom

© Higher Education Press and Springer-Verlag 2009

**Abstract** Many software engineering tasks can potentially be automated using search heuristics. However, much work is needed in designing and evaluating search heuristics before this approach can be routinely applied to a software engineering problem. Experimental methodology should be complemented with theoretical analysis to achieve this goal. Recently, there have been significant theoretical advances in the runtime analysis of evolutionary algorithms (EAs) and other search heuristics in other problem domains. We suggest that these methods could be transferred and adapted to gain insight into the behaviour of search heuristics on software engineering problems while automating software engineering.

**Keywords** software engineering, evolutionary algorithms, runtime analysis

## 1 Introduction

Software development is expensive. One strategy for reducing development costs is to identify software engineering tasks that can be automated. The field of search based software engineering (SBSE) suggests that automation can be achieved by reformulating the software engineering tasks as optimisation problems to which search heuristics can be applied. In principle, the only required ingredients in the approach is an encoding of candidate solutions and a way of comparing the quality of two candidate solutions.

Why should search heuristics sometimes be preferred over classical algorithms? Many software engineering problems are NP-hard [1], hence it is unlikely that they can be solved exactly with efficient algorithms. Although one cannot in general guarantee that search heuristics provide optimal solutions, it has been found that search heuristics often give satisfactory results on several NP-hard problems. Hence, it is worth also considering search heuristics on hard software engineering problems. Search heuristics can also be attractive on computationally tractable problems, e.g., when there are insufficient human resources to develop a problem-specific algorithm. Search heuristics can often be adapted with relatively little effort to new problem domains with satisfactory results [2].

In fact, search-based software engineering has already proven successful in several software engineering domains. A large body of the literature has focused on software testing [3], but there is a long list of other software engineering problems which have been successfully tackled with search heuristics. For an overview of some of this work, see Clark et al. [4], and Harman [1].

Although the application of search heuristics in principle reduces many software engineering problems to finding an appropriate fitness function and solution encoding, it also introduces some new methodological challenges. The theoretical understanding of these algorithms is incomplete. Despite several decades of research on search heuristics, many fundamental questions regarding their behaviour remain open and the theoretical understanding of their behaviour is incomplete. Hence, when experiments sometimes show that a given search heuristic has an unsatisfactory performance on a problem, the underlying reasons for this are many times unclear. Sometimes, the successful application of a search heuristic on a given problem depends on finding appropriate parameter settings, and the cost of finding such parameter settings can be high. Good experimental design can in some cases reduce the cost of parameter tuning [5]. How-

Received September 8, 2008; accepted October 29, 2008

E-mail: P.K.Lehre@cs.bham.ac.uk

ever, finding the correct parameter setting may be futile if the algorithm is unsuitable for the problem. In general, there does not exist a single search heuristic that works optimally on all problems [6]. Hence, there is a need to better understand which search heuristics are most applicable to a given software engineering problem.

Theoretical research like runtime analysis will seldom provide the optimal choice of search heuristic or parameter settings for specific real world problems. However, it may provide insight into how and why search heuristics work and sometimes fail. In particular, a theoretical analysis can point out simple general cases where the choice of an algorithm or parameter setting has a particularly important effect on the behaviour of the algorithm. With an understanding of how search heuristics behave in such archetypical cases, a practitioner will be better equipped for making an informed decision as to how to choose parameter settings and heuristics on a specific real world problem.

So far, there has been little theoretical research in search-based software engineering. Except for the research related to runtime analysis of search heuristics that will be described in Sections 3 and 4, the only theoretical study within evolutionary computation on SBSE we are aware of is the work by Harman and McMinn [7] where schema theory is adapted to the evolutionary software testing domain. Their theoretical study suggests that there exist so-called Royal Road problems in evolutionary testing where genetic algorithms using crossover should have an advantage over hill climbers.

The rest of this paper is organised as follows. Section 2 introduces theoretical runtime analysis and briefly reviews methods and results that have been obtained for randomised search heuristics in general problem domains. The next two sections focus on runtime results that have been obtained specifically for software engineering problems. These results are related to problems in conformance testing of finite state machines and structural software testing. Section 5 suggests future directions and open problems for runtime analysis within SBSE. Finally, the paper is concluded in Section 6.

## 2 Theoretical research in SBSE

### 2.1 Runtime analysis

Evolutionary algorithms and other randomised search heuristics are attractive due to their versatility. However, in contrast to many problem-specific algorithms, it can be notoriously difficult to establish exactly how these algorithms work, and why they sometimes fail. Empirical investigations can be costly and do not always yield the desired level of information needed to make the right choice of heuristic and corresponding parameter setting at hand.

To put the application of search heuristics in software engineering and other domains on a firmer ground, it is desirable to construct a theory which can explain the basic principles of the heuristics and possibly provide guidelines for developing new and improved algorithms. Such a theory should preferably be valid without making simplifying assumptions about the algorithms or problems, e.g., assuming that the EA has infinite population size, or ignoring the stochastic nature of these algorithms.

When studying a particular search heuristic, it is important that one makes clear what class of problems one has in mind. One can say very little about the advantages and disadvantages of a heuristic without making any assumption about the problem [6].

For a given heuristic and problem class, an initial theoretical question to ask, is whether the heuristic will ever find a solution, if it is allowed unlimited time. This type of question falls within the realms of convergence analysis, which is a well-developed area [8]. There exist simple conditions on the underlying Markov chain of a search heuristic that guarantee convergence in finite time. These conditions often hold for the popular heuristics [8]. Note that convergence itself gives very little information about whether an algorithm is practically useful, because no limits are put on the amount of resources the algorithm uses.

If convergence can be guaranteed within unlimited time, the next question to ask is how much time the heuristic needs to find the solution. This type of question falls within the realms of runtime analysis of search heuristics, and requires a measure of time. Time can be measured as the number of basic operations in the search heuristic. Usually, the most time-consuming operation in an iteration of a search algorithm is the evaluation of the cost function. We therefore adopt the *black-box scenario* [9], in which time is measured as the number of times the algorithm evaluates the cost function.

**Definition 2.1** (Runtime [10,11]) Given a class $\mathcal{F}$ of cost functions $f_i : S_i \to \mathbb{R}$, the runtime $T_{A,\mathcal{F}}(n)$ of a search algorithm $A$ is defined as

$$T_{A,\mathcal{F}}(n) := \max\{T_{A,f} \mid f \in \mathcal{F} \text{ with } \ell(f) = n\},$$

where $\ell(f)$ is the problem instance size, and $T_{A,f}$ is the number of times algorithm $A$ evaluates the cost function $f$ until the optimal value of $f$ is evaluated for the first time.

A typical search algorithm $A$ in the black-box scenario is randomised. Hence, the corresponding runtime $T_{A,\mathcal{F}}(n)$ will be a random variable. The runtime analy-

sis will therefore seek to estimate properties of the distribution of the random variable $T_{A,\mathcal{F}}(n)$, in particular the *expected runtime* $\mathbf{E}[T_{A,\mathcal{F}}(n)]$ and the *success probability* $\mathbf{Pr}[T_{A,\mathcal{F}}(n) \leqslant t(n)]$ for a given time bound $t(n)$.

The last decade of research in the area show that it is important to apply appropriate mathematical techniques to get good results [12]. Initial studies of exact Markov chain models of search heuristics were not fruitful, except for the the simplest cases. A more successful and particularly versatile technique has been the so-called drift analysis [11,13], where one introduces a potential function which measures the distance from any search point to the global optimum. By estimating the expected one-step drift towards the optimum with respect to the potential function, one can deduce bounds on the expected runtime and the success probability. Finding the right potential function can sometimes be a challenge, and can differ considerably from the objective function. In addition to drift analysis, the wide range of techniques used in the study of randomised algorithms [14], in particular Chernoff bounds, have proved useful also for evolutionary algorithms.

Initial runtime studies were concerned with simple EAs like the (1+1) EA on artificial pseudo-boolean functions [10, 15,16]. These studies established fundamental facts about the (1+1) EA, e.g., that it can optimise any linear function in $O(n \log n)$ expected time [10], that quadratic functions with negative weights are hard [16], that the hardest functions require $\Theta(n^n)$ iterations [10] and, in contrast to commonly held belief, that not all unimodal functions are easy [15]. The understanding of the runtime of search heuristics was then expanded in several directions, by analysing parameter settings (e.g., the crossover operator [17,18], population size [19, 20] and diversity mechanisms [21,22]), by analysing new algorithms (e.g., ant colony optimisation [23] and particle swarm optimisation [24]), and by considering new problem settings (e.g., multi-objective [25, 26] and continuous [27] optimisation).

The analysis of search heuristics then shifted its focus from artificial pseudo-boolean functions to classical combinatorial optimisation problems. Many of these results are covered in the survey [28]. Initially, combinatorial optimisation problems in P were analysed [29−32]. Giel and Wegener analysed maximum matching, showing that although the runtime of (1+1) EA is in general exponential, the EA is a polynomial-time randomised approximation scheme (PRAS) for the problem [29]. Other problems analysed include sorting [30], minimum spanning tree [31], and Eulerian cycle [32]. For NP-hard problems, one must expect that the worst-case expected runtime is exponential. Hence, the focus has been on analysing the runtime on interesting subclasses, e.g., for the vertex cover problem [33], or on the average case runtime over the set of instances [34], or the approximation quality that can be obtained by the algorithm in polynomial time [34, 35].

### 2.2 Runtime analysis of search heuristics for software engineering

Runtime analysis is practically unexplored within search based software engineering. This might be due to the fact that many of these problems have been outside the reach of the analysis techniques available, partly because many software engineering problems are NP-hard [1]. The results on conformance testing of finite state machines and structural software testing that will be presented in the next two sections are the first attempts in theoretical analysis of the runtime of evolutionary algorithms in search based software engineering.

## 3 Conformance testing of finite state machines

The behaviour of many software systems can be described in terms of finite state machines (FSMs). For example, a server conforming to a communication protocol, can in principle be described as a system that is in some internal state, and which on external inputs outputs a response and makes a state transition.

Black box testing of finite state machines consists of deducing properties about a finite state machine from its input output behaviour [36]. Conformance testing is a type of black box testing where the objective is to determine whether the black box FSM is identical to a specification FSM whose internals are known. One of the challenges in conformance testing is the state verification problem [37], which consists of determining for a given state, whether the black box FSM is in this state or not. The state verification problem can be solved using unique input output (UIO) sequences. A UIO sequence for a given state $s$ in an FSM is an input sequence $x$ which causes the FSM to output a sequence $y$ if and only if the FSM was initially in state $s$.

UIOs are useful in FSM testing, and it would be desirable to have methods which compute them efficiently. Unfortunately, such exact and efficient methods are ruled out due to the NP-hardness of the UIO problem [37]. In certain cases, the UIOs are sufficiently short to be found through exhaustive or random search. However, one can hypothesise that more sophisticated search heuristics will be more efficient.

Experimental results show that EAs can construct UIOs efficiently on some instances. Guo et al. compared an evo-

lutionary approach with a random search strategy, and found that the two approaches have similar performance on a small FSM, while the evolutionary approach outperforms random search on a larger FSM [38]. Derderian et al. presented an alternative evolutionary approach [39], confirming Guo et al.'s results.

Theoretical runtime analysis confirm that EAs can outperform random search on the UIO problem [40]. The expected running time of the (1+1) EA on a counting FSM instance class is $O(n \log n)$, while random search needs exponential time [40]. The UIO problem is NP-hard [36], so one can expect that there exist EA-hard instance classes. It has been proved that a combination lock FSM class is hard for the (1+1) EA [40]. To reliably apply EAs to the UIO problem, it is necessary to distinguish easy from hard instances. Theoretical results indicate that there is no sharp boundary between these categories in terms of runtime. For any polynomial $n^k$, there exist UIO instance classes, as shown in Fig. 1, where the (1+1) EA has running time $\Theta(n^k)$ [40].

These theoretical results demonstrate that the runtime of the (1+1) EA on the UIO problem is highly dependent on the characteristics of the FSM. A later study investigated to what degree the runtime depends on characteristics of the EA itself [41]. In particular, the study focused on the impact of the acceptance criterion in the (1+1) EA, and on the effect of a population and crossover in a steady state genetic algorithm.

With regards to the choice of the acceptance criterion, the RIDGE FSM instance class is described which induces a search space with a neutral path of equally fit search points. Runtime analysis shows that the variant of (1+1) EA which only accepts strictly better search points will get stuck on the path, while the standard (1+1) EA which also accepts equally fit search points will find the UIO in polynomial time. This result shows that apparently minor modifications of an algorithm can have an exponentially large runtime impact when computing UIOs.

Regarding the role of populations and crossover on the UIO problem, the runtime of the ($\mu$+1) steady state genetic algorithm is analysed. The result shows that on the TWOPATHS FSM instance class shown in Fig. 2, the SSGA finds the UIO in polynomial time as long as the crossover probability is a non-zero constant and the population is sufficiently large. However, with crossover probability 0, the runtime of the ($\mu$+1) SSGA increases exponentially. The analysis is based on the observation that the fitness function corresponding to the TWOPATHS FSM consists of two separate paths, each leading to a local optimum. The global optimum is located between these two local optima. The individuals in the population can follow either of the paths, eventually leading to two sub-populations, one on each local optimum. The paths are constructed in such a way that it is unlikely that mutation alone will produce the global optimum. However, the global optimum can easily be reached by crossing over two individuals that sit on different optima. It is worth noting that this analysis is based on specific types of crossover and mutation. This result means that when computing UIOs, the crossover operator can be essential, and simple EAs including the (1+1) EA can be ineffective. This result is important because although the crossover operator is often thought to be important for GAs, there exist very few theoretical results on non-artificial problems confirming that this is the case.

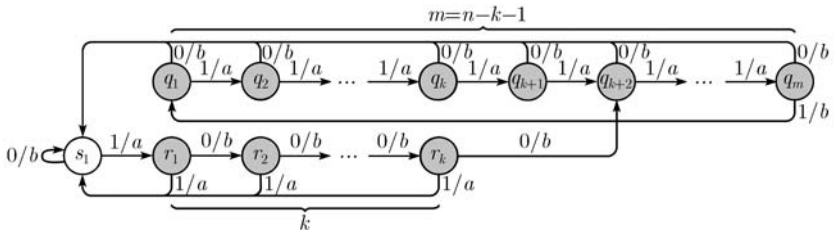


**Fig. 1** Finding a UIO for state $s_1$ with (1+1) EA becomes harder when increasing parameter $k$

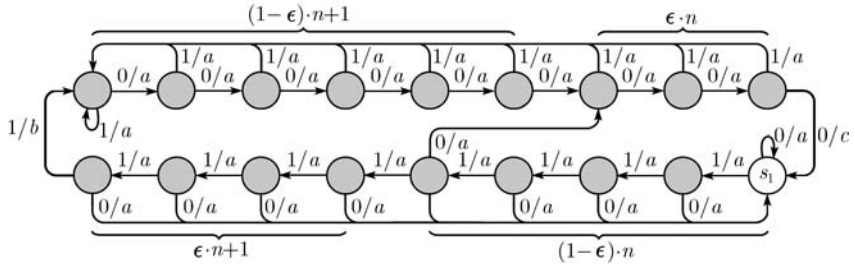


**Fig. 2** TWOPATHS FSM instance class where crossover is useful

## 4 Structural software testing

Software testing is both an important and costly task in software development. Much research in SBSE has therefore focused on automating software testing using search heuristics [3]. Structural testing is a particular type of software testing where test cases are generated using knowledge about the internal structure of the program. The goal is to generate test cases which make different parts of the program become executed.

Arcuri et al. [42] analysed the runtime of three search heuristics on generating test data for the triangle classification program, which is a program that occurs frequently in the analysis of testing approaches. The program consists of a single function which takes three integers `x`, `y` and `z` as arguments. The task of the program is first to determine whether there exists a triangle with side lengths `x`, `y` and `z`, and secondly, if such a triangle exists, return the triangle type. The possible triangle types are scalene, equilateral, and isosceles.

The test generation task for a given program branch, is to find input values `x`, `y`, and `z` such that the branch is executed. Whether a given branch is executed may depend on whether predicates in conditional statements that occur in the program are satisfied by the program variables. In the case of the equilateral branch, it is necessary that the program variables satisfy the predicates `a+b>c` and `a=b=c`. Empirical investigations have shown that the equilateral branch is the hardest branch to generate test cases for. Therefore, the runtime analysis focused on the problem of finding integers `x`, `y` and `z` such that the equilateral branch is executed by the program.

```
int tri_type(int x, int y, int z) {
  int type;
  int a=x, b=y, c=z;
  if (x > y) {
     //TARGET 2
     int t = a; a = b; b = t;
  }
  if (a > z) { int t = a; a = c; c = t; }
  if (b > c) { int t = b; b = c; c = t; }
  if (a + b <= c) {
      type = NOT_A_TRIANGLE;
  } else {
      type = SCALENE;
      if (a == b && b == c) {
          // TARGET 1
          type = EQUILATERAL;
      } else if (a == b || b == c) {
          type = ISOSCELES;
      }
    }
  return type;
}
```

**Fig. 3** Triangle classification (TC) program, adapted from [3]

Search based test case generation techniques are often guided by a branch distance measure [3]. The branch distance gives an indication as to how "close" the program variables are to satisfy the predicates that must be satisfied to execute the branch. The particular branch distance function depends on the type of nested conditional statements that occur before the branch. As an example, for the single conditional statement `if(a>b)`, the branch distance measure is $\max\{0, \texttt{b}-\texttt{a}\}$.

In order to analyse the asymptotic runtime on this problem, it is necessary to specify the meaning of the meaning of instance size. The program structure itself is a constant. In this study, the instance size is defined in terms of the size $n$ of the integer range $[-n/2, \ldots, n/2-1]$ in which the variables `x`, `y` and `z` are searched for. Other definitions of instance size may be more suitable on different formulations of the test data generation problem.

The first and simplest search heuristic considered search (RS), which in each iteration samples three integers in the chosen interval uniformly at random until a correct test is found [42]. The probability of finding the optimal solution is therefore the same in each iteration, and the expected runtime is and the expected runtime is $\Theta(n^2)$.

The second method considered is a hill climber (HC) the first fitter neighbour in the unit neighbourhood according to the branch distance measure, or restarts from a random point if no such neighbour exists. The expected distance to the optimum in the initial iteration is linear with respect to the size of the interval range, and the maximal reduction in distance to the optimum per iteration is one. By showing that the expected number of restarts is constant, one arrives at the expected runtime $\Theta(n)$ [42]. $\Theta(n)$ [42].

The third method is called the alternating variable which works similar to the hill climber, but increasingly large step sizes as long as a fitter neighbour solution is found. Hence, this algorithm is able to overcome longer distances than HC per iteration. The expected runtime of AVM is bounded from below by $\Omega(\log n)$ and from above by $O((\log n)^2)$ [42].

## 5 Future directions

### 5.1 Relationships between SE problems and search heuristics

The range of software engineering (SE) problems for which search based approaches could be considered is wide [4]. Problem characteristics can vary greatly between different problems. Hence, a search heuristic which is suitable for one

problem may be unsuitable for another problem. A central goal with runtime analysis of search heuristics on software engineering problems is to develop a theoretical foundation for understanding the relationships between software engineering problems and search heuristics. Such an understanding will aid in choosing an appropriate search heuristic for a given software engineering problem. The following subsections describe directions for research that when taken would lead the field closer to reaching this goal.

### 5.2 Better mathematical techniques

The recent progress in the runtime analysis of search heuristics is partly due to the application of appropriate mathematical techniques [11, 12]. The exact Markov chain models of evolutionary algorithms that were considered in early studies often became intractable in more complex scenarios.

It is possible that further progress in the field depends on finding and adopting mathematical techniques that have not been previously considered in the field. In particular, new techniques may be required to analyse search heuristics that have not previously been investigated theoretically. For example, population-based evolutionary algorithms may behave qualitatively differently than the (1+1) EA on exactly the problems for which they have an advantage over the (1+1) EA.

New theoretical approaches may also be helpful when considering broader instance classes. In the studies described in Sections 3 and 4, very detailed information about the problem structure was needed to arrive at good bounds. Better techniques may allow good bounds to be obtained with less detailed information about the problem structure.

### 5.3 Analysis of more complex search heuristics

Most runtime analyses so far have considered the (1+1) EA, and the understanding of this algorithm has advanced quite far. The (1+1) EA can be classified as a simple search heuristic because it only considers a single individual that in each step is updated by a mutation operator. More complex search heuristics include those that consider populations and a crossover operator (e.g., genetic algorithms), probability distributions over the search space (estimation of distribution algorithms), several optimisation objectives (multi-objective EAs) and multiple populations (e.g., co-evolutionary algorithms).

There exist relatively few results about the runtime of more complex evolutionary algorithms that employ a population of individuals and a crossover operator. Population-based evolutionary algorithms tend to be harder to analyse, partly due to the complex interactions that can occur between individuals in the population and the way the crossover operator exchanges information between individuals. It is difficult to capture the stochastic behaviour of the entire population using a simple potential function. Furthermore, such algorithms often have more parameters, and the settings of these may have important effects on the runtime behaviour.

Although the (1+1) EA has been shown to be surprisingly efficient in many problem domains, there exists important cases where this algorithm is inefficient [41] and where it is known that the crossover operator and a population can help. Furthermore, these complex algorithms tend to be used more often by practitioners. It is therefore important to extend the runtime analysis to more complex evolutionary algorithms.

Although estimation of distribution algorithms (EDAs) have gained significant popularity in recent years, little is known about the asymptotic runtime behaviour of these algorithms. One exception is the result by Droste [43] which shows that the expected runtime of cGA on the class of linear functions is $\Theta(nK)$, where $K = n^{1+\epsilon}, \epsilon > 0$ is a parameter in the algorithm. Furthermore, Chen et al. [44] showed that the incremental UMDA is efficient on the LEADINGONES problem, and also constructed a problem for which the incremental UMDA is inefficient. There exists no theoretical result about the runtime of any EDA on any software engineering problem. A useful contribution in this area would be to identify a software engineering problem for which the particular characteristics of EDAs make this algorithm more efficient than traditional evolutionary algorithms. Sagarna et al. [45] suggest that the probability distribution which the EDA learns during the optimisation process may contain useful information about problem characteristics.

Many software engineering problems involve multiple conflicting objectives, and in such cases it is inadequate to optimise the solution considering a single objective only. Examples include problems where it is necessary to balance the cost and the value of a solution. Multi-objective optimisation problems can be tackled using techniques from evolutionary multi-objective optimisation [46] which have proved successful in finding sets of Pareto optimal solutions.

The multi-objective point of viewing optimisation is gaining popularity within search based software engineering. Applications include the testing-resource allocation problem [47], test data generation [48], test case selection [49] and requirements engineering [50].

The runtime of some multi-objective evolutionary algorithms (MOEA) have been analysed on simple functions [20, 51, 52], and some multi-objective combinatorial optimisation problems [53]. However, so far, there has been no theoretical analysis of evolutionary multi-objective algorithms in the software engineering domain. Although difficult, it is likely that such an analysis would be beneficial. In partic-

ular, a rigorous mathematical analysis may provide insight into the structure and connectivity of the Pareto set of various software engineering problems.

Interestingly, co-evolutionary techniques have been used to to co-evolve programs and tests [54]. Although it has been shown that it is possible to analyse the runtime of co-evolutionary algorithms [55], it may not be feasible to analyse the runtime in this particular application. However, in the more general case, theoretical runtime analysis could potentially provide insight into some of the challenges that occur in the application of co-evolution, including disengagement, cycling, mediocre stable states and forgetting.

### 5.4 Analysis of broader problem classes

The existing theoretical analyses in SBSE described in Sections 3 and 4 consider restricted instance classes of software engineering problems. It would be a valuable contribution to generalise these results to broader instance classes. One possible approach is to seek easily computable problem instance characteristics with which instances that are hard for a given search heuristic can be identified. Such results could allow practitioners to quickly rule out a given search heuristic when it can be proved that it is unsuitable for the problem at hand.

Many software engineering problems are special cases of classical combinatorial optimisation problems. A line of research could be to investigate whether the real world instances of a given software engineering problem share some special properties. If all the real world instances belong to a particular subclass of the instances, then it would be useful to analyse the runtime of a search heuristic with respect to this class of instances. For some NP-hard problems that occur in software engineering, one can hope that the real world instances of the problem form tractable subclasses of the problem.

Much of the theoretical research in SBSE is related to software testing problems. However, SBSE approaches have been applied to a much wider range of software engineering problems. Many of these problems are special cases of classical combinatorial optimisation problems for which there exists an extensive body of theoretical research. Runtime analysis of search heuristics could potentially benefit from this research, and provide useful hints about problem structure. An example where existing knowledge about problem structure has been used occurs in Giel and Wegener's analysis of (1+1) EA for the maximum matching problem, where the idea of augmenting paths helps to understand the behaviour of the algorithm on the problem [29].

### 5.5 Approximation quality of search heuristics

In some cases, it is not required that the solution to a software engineering problem is the globally best solution, as long as it is sufficiently good according to some criteria. In such cases, it is desirable to acquire knowledge about the approximation quality that can be obtained by the search heuristic within polynomial time. If one can prove that for any problem instance, the solution found by a search heuristic is always within a certain bound of the optimal solution, then this would in some cases be a sufficient certificate that a search heuristic is useful on the problem. Results on the approximation quality of search heuristics also provide valuable information for choosing between two heuristics. There exist currently no results about the approximation quality of search heuristics in the software engineering domain.

## 6 Conclusion

Search based software engineering is a promising approach to automating certain software engineering tasks. In this paper, we have pointed out some methodological challenges that are involved in the experimental evaluation of search heuristics on software engineering problems and suggest that empirical methodologies should be complemented with a theoretical foundation. We describe some of the progress that has been made in runtime analysis of search heuristics applied in other problem domains and suggest that this type of theoretical analysis is also applicable in search based software engineering. We then describe recent results on the runtime of search heuristics in conformance testing of finite state machines, and in structural testing of software. Finally, we propose several avenues for further research on runtime analyses in the area of search based software engineering.

**Acknowledgements** The authors would like to thank Pietro Oliveto, Ramon Sagarna, Andrea Arcuri and the other members of the SEBASE project† for useful comments. This work was partially supported by EPSRC (EP/C520696/1) and by the Royal Society under a grant in its UK-China Science Network programme.

## References

1. Harman M. The current state and future of search based software engineering. In: Proceedings of International Conference on Software Engineering / Future of Software Engineering 2007 (ICSE/FOSE 2007), IEEE Computer Society, 2007, 342–357
2. Sarker R, Mohammadian M, Yao X, eds. Evolutionary Optimization. Kluwer Academic Publishers, 2002

† Softhare engineering by automated search (SEBASE) project web site. http://www.cercia.ac.uk/projects/research/SEBASE/

3. McMinn P. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 2004, 14(2): 105–156
4. Clark J A, Dolado J J, Harman M, Hierons R M, Jones B, Lumkin M, Mitchell B, Mancoridis S, Rees K, Roper M, Shepperd M. Reformulating software engineering as a search problem. IEEE Proceedings-Software, 2003, 150(3): 161–175
5. Poulding S, Emberson P, Bate I, Clark J A. An efficient experimental methodology for configuring search-based design algorithms. In: Proceedings of 10th IEEE High Assurance System Engineering Symposium (HASE'2007), 2007, 53–62
6. Wolpert D H, Macready W G. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 67–82
7. Harman M, McMinn P. A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation. In: Proceedings of the ISSTA 2007 Symposium, 2007, 73–84
8. Rudolph G. Finite markov chain results in evolutionary computation: A tour d'horizon. Fundamenta Informaticae, 1998, 35(1): 67–89
9. Droste S, Jansen T, Wegener I. Upper and lower bounds for randomized search heuristics in black-box optimization. Theory of Computing Systems, 2006, 39(4): 525–544
10. Droste S, Jansen T, Wegener I. On the analysis of the (1+1) Evolutionary Algorithm. Theoretical Computer Science, 2002, 276: 51–81
11. He J, Yao X. A study of drift analysis for estimating computation time of evolutionary algorithms. Natural Computing, 2004, 3(1): 21–35
12. Wegener I. Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In: Sarker R, Mohammadian M, Yao X, eds. Evolutionary Optimization. Dordrecht: Kluwer, 2002, 349–369
13. Oliveto P, Witt C. Simplified drift analysis for proving lower bounds in evolutionary computation. In: Proceedings of Parallel Problem Solving from Nature (PPSN'X). Berlin: Springer, LNCS, 2008, 5199: 82–91
14. Motwani R, Raghavan P. Randomized Algorithms. Cambridge: Cambridge University Press, 1995
15. Droste S, Jansen T, Wegener I. On the optimization of unimodal functions with the (1+1) evolutionary algorithm. In: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN'V). London: Springer-Verlag, 1998, 13–22
16. Wegener I, Witt C. On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. Journal of Discrete Algorithms, 2005, 3(1): 61–78
17. Jansen T, Wegener I. Real royal road functions–where crossover provably is essential. Discrete Applied Mathematics, 2005, 149(1-3): 111–125
18. Storch T, Wegener I. Real royal road functions for constant population size. Theoretical Computer Science, 2004, 320(1): 123–134
19. Witt C. Population size versus runtime of a simple evolutionary algorithm. Theoretical Computer Science, 2008, 403(1): 104–120
20. Giel O, Lehre P K. On the effect of populations in evolutionary multi-objective optimization. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO'06). New York: ACM, 2006, 651–658
21. Friedrich T, Oliveto P S, Sudholt D, Witt C. Theoretical analysis of diversity mechanisms for global exploration. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08). New York: ACM, 2008, 945–952
22. Friedrich T, Hebbinghaus N, Neumann F. Rigorous analyses of simple diversity mechanisms. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO'07), 2007, 1219–1225
23. Neumann F, Witt C. Runtime analysis of a simple ant colony optimization algorithm. In: Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC'2006). Berlin: Springer, LNCS, 2006, 4288: 618–627
24. Sudholt D, Witt C. Runtime analysis of binary pso. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08). New York: ACM, 2008, 135–142
25. Giel O. Zur Analyse von randomisierten Suchheuristiken und Online-Heuristiken. PhD thesis. Dortmund: Universität Dortmund, 2005
26. Neumann F. Combinatorial Optimization and the Analysis of Randomized Search Heuristics. PhD thesis. Kiel: Christian-Albrechts-Universität zu Kiel, 2006
27. Jägersküpper J. Probabilistic Analysis of Evolution Strategies Using Isotropic Mutations. PhD thesis. Dortmund: Universität Dortmund, 2006
28. Oliveto P S, He J, Yao X. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. International Journal of Automation and Computing, 2007, 4(1): 100–106
29. Giel O, Wegener I. Evolutionary algorithms and the maximum matching problem. In: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2003), 2003, 415–426
30. Scharnow J, Tinnefeld K, Wegener I. Fitness landscapes based on sorting and shortest paths problems. In: Proceedings of 7th Conference on Parallel Problem Solving from Nature (PPSN–VII). Berlin: Springer, LNCS, 2002, 2439: 54–63
31. Neumann F, Wegener I. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theoretical Computer Science, 2007, 378(1): 32–40
32. Doerr B, Klein C, Storch T. Faster evolutionary algorithms by superior graph representation. In: Proceedings of the 1st IEEE Symposium on Foundations of Computational Intelligence (FOCI'2007), 2007, 245–250
33. Oliveto P, He J, Yao X. Analysis of population-based evolutionary algorithms for the vertex cover problem. In: Proceedings of IEEE World Congress on Computational Intelligence (WCCI'08), 2008, 1563–1570
34. Witt C. Worst-case and average-case approximations by simple randomized search heuristics. In: Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'05), LNCS, 2005, 3404: 44–56
35. Friedrich T, Hebbinghaus N, Neumann F, He J, Witt C. Approximating covering problems by randomized search heuristics using multi-objective models. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO'07). New York: ACM Press, 2007, 797–804
36. Lee D, Yannakakis M. Principles and methods of testing finite state machines-a survey. In: Proceedings of the IEEE, 1996, 84(8): 1090–1123
37. Lee D, Yannakakis M. Testing finite-state machines: state identification and verification. IEEE Transactions on Computers, 1994, 43(3): 306–320
38. Guo Q, Hierons R M, Harman M, Derderian K A. Computing unique input/output sequences using genetic algorithms. In: Proceedings of the 3rd International Workshop on Formal Approaches to Testing of

Software (FATES'2003), LNCS, 2004, 2931: 164–177
39. Derderian K A, Hierons R M, Harman M, Guo Q. Automated unique input output sequence generation for conformance testing of fsms. The Computer Journal, 2006, 49(3): 331–344
40. Lehre P K, Yao X. Runtime analysis of (1+1) EA on computing unique input output sequences. In: Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC'07). IEEE Press, 2007, 1882–1889
41. Lehre P K, Yao X. Crossover can be constructive when computing unique input output sequences. In: Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL'08), LNCS, 2008, 5361: 595–604
42. Arcuri A, Lehre P K, Yao X. Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem. In: Proceedings of the 1st International Workshop on Search-Based Software Testing, 2008, 161–169
43. Droste S. A rigorous analysis of the compact genetic algorithm for linear functions. Natural Computing, 2006, 5(3): 257–283
44. Chen T, Tang K, Chen G, Yao X. On the analysis of average time complexity of estimation of distribution algorithms. In: Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC'07). IEEE Press, 2007, 453–460
45. Sagarna R, Arcuri A, Yao X. Estimation of distribution algorithms for testing object oriented software. In: Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC'07). 2007, 438–444
46. Deb K. Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, 2001
47. Wang Z, Tang K, Yao X. A multi-objective approach to testing resource allocation in modular software systems. In: Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC'08). IEEE Press, 2008, 1148–1153
48. Harman M, Lakhotia K, McMinn P. A multi-objective approach to search-based test data generation. In: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO'07). ACM, 2007, 1098–1105
49. Yoo S, Harman M. Pareto efficient multi-objective test case selection. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07). ACM, 2007, 140–150
50. Zhang Y, Harman M, Mansouri S A. The multi-objective next release problem. In: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07). ACM, 2007, 1129–1137
51. Laumanns M, Thiele L, Zitzler E. Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. IEEE Transactions on Evolutionary Computation, 2004, 8(2): 170–182
52. Giel O. Expected runtimes of a simple multi-objective evolutionary algorithm. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC'03). IEEE Press, 2003, 3: 1918–1925
53. Neumann F. Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. In: Proceedings of Parallel Problem Solving from Nature (PPSN'VIII), 2004, 81–90
54. Arcuri A, Yao X. Coevolving programs and unit tests from their specification. In: Proceedings of IEEE International Conference on Automated Software Engineering (ASE), 2007, 397–400
55. Jansen T, Wiegand R P. The cooperative coevolutionary (1+1) EA. Evolutionary Computation, 2004, 12(4): 405–434