Realism in Statistical Analysis of Worst Case Execution Times

D. Griffin[1] and A. Burns[1]

Abstract

This paper considers the use of Extreme Value Theory (EVT) to model worst-case execution times. In particular it considers the sacrifice that statistical methods make in the realism of their models in order to provide generality and precision, and if the sacrifice of realism can impact the safety of the model. The Gumbel distribution is assessed in terms of its assumption of continuous behaviour and its need for independent and identically distributed data. To ensure that predictions made by EVT estimations are safe, additional restrictions on their use are proposed and justified.

1. Introduction

By their nature, hard real time systems have deadlines to meet, with consequences for failure. This leads to the problem of worst case execution time: finding out the minimum bound on a program's runtime. Unfortunately, in general this is an unanswerable question, as solving this would solve the halting problem. If restrictions are enforced on the program, then the question may be answerable - however, it may not be tractable.

The nature of these restrictions can be understood by noting that WCET estimation involves an element of model building, and therefore the work of Levins [10] is applicable. Levins describes a three-fold trade-off on useful models between *generality*, *realism* and *precision*, and argues that no useful model can maximize all three of these properties. Whilst Levins did not define these terms, assuming the meanings to be obvious, they have come to mean the following:

- *Generality*: The degree to which the model is applicable to multiple situations in the real world.
- *Realism*: The degree to which the model accurately represents the actual phenomena occurring in the real world.
- *Precision*: The degree to which error bounds are minimised when comparing the model's predictions with the real world.

The term "useful model" was later defined as one which is understandable, measurable, and computationally soluble [12]. Later work by Bullock and Silverman [3] named these three properties the models *tractability*, and constructed an argument for a four-fold tradeoff between tractability and the original properties of a useful model defined by Levins. Within this four-fold tradeoff, Levins useful models are situated at the threshold between intractable and tractable models.

[1]Department of Computer Science, University of York, UK.

Levins' philosophy is demonstrated in current approaches to WCET estimation such as abstract interpretation [7] which sacrifices precision to gain an approach which is general and realistic - and the lack of precision manifests itself in a relatively high overestimation of the true WCET. Unfortunately, the requirement to guarantee the safety of the estimation, that the WCET is bounded from above, not below, requires an element of precision. Further, current research into abstract interpretation is increasing the precision of the method, which according to Levins will require another aspect to be sacrificed. One present approach achieves this by sacrificing some realism by using a heuristic [14]. Other approaches may sacrifice generality, or may simply build a more precise and more complicated model which sacrifices tractability.

Statistical analysis, as proposed by Edgar and Burns [5, 6] is a method which takes a radically different approach to abstract interpretation. Statistical analysis sacrifices realism for generality and precision, by using extreme value theory (EVT) statistics [9]. EVT is a collection of statistical models which are suited to accurately predicting the tail end of a distribution. By fitting one such model, the Gumbel distribution, to observed data the probability of failing to meet a given execution time can be found, and this can be used as the foundation of a probabilistic real time system [2]. Clearly, EVT is not a realistic model of a computer system, but it can model observed behavior precisely. Statistical analysis was studied further by Hansen, Hissam and Moreno [8], who modified the work to fit more accurately with the original design of EVT. The same modifications also produce results in the form of probability of failure in a given time period, as preferred by the testing community [4].

This paper aims to look at an issue which has not yet been adequately addressed: the impact of the lack of realism in the model on the safety of Gumbel-derived estimates. Primarily, this manifests itself in subtle differences between the Gumbel distribution and the properties of an actual system. Sections 2 and 3 detail two significant lapses of reality in the model, and demonstrate how unsafe predictions could be made using statistical analysis. Section 4 details methods to take into account these lapses of reality, either by proving that they do not apply or adapting the application of statistical analysis to compensate. Conclusions are presented in Section 5.

2. Continuous vs. Discrete Distributions

The Gumbel distribution, along with any other continuous probability distribution, makes an assumption that all values are possible. This is clearly unrealistic; looking at the control flow graph of a program will show that a program can not terminate at any point. Instead, programs perform computation for a period of time, and then may stop or perform more computation. For instance, consider the program in Figure 1, being run on a simple processor with no features to speed up execution. The function F refines a result until either the result is accurate enough or the routine hits a recursion depth limit. This program may or may not exhibit exponential decay in the probability of its execution times depending on the nature of G, the refinement function, and the input x. However, it has the property that it cannot terminate during an execution of G, or the accuracy test. This means that the only times it can terminate are the times expressible as:

$$\sum_{i=0}^{j} rt(f(x_i)) \text{ where } j \in [0, 5], \text{ and } x_i \in Dom(F)$$

Assuming that the domains of the functions F and G are finite, then the program may only terminate

```
F(x, d)
1  x = G(x)
2  if x has sufficient accuracy or d ≥ 5
3      return x
4  else
5      return F(x, d + 1)
```

Figure 1. A program which does not have a continuous distribution of runtimes

at finitely many times. Further, the set of all possible termination times need not occupy the entire interval of termination times. For instance, F and G may be written so the program takes a multiple of 5 cycles to complete. Then the program will not terminate on any cycle which isn't a multiple of 5.

This becomes a problem because the Gumbel distribution assumes that the program can stop at any point, and hence will produce results which do not make sense. Using the previous example, it is known that the probability of termination by cycle 9 is the same as termination by cycle 6 - if the program hasn't terminated before cycle 6, it cannot terminate until cycle 10. The prediction of the Gumbel distribution differs, as shown in Figure 2. In fact, the Gumbel distribution produces values lower than the actual distribution - and these values are unsafe.

Hence the problem is as follows. By using a continuous function to model the execution time of the program, an assumption is made on the program that it can terminate at any point. Whilst any synchronous processor has some form discrete time, therefore invalidating this assumption, such small errors would pass unnoticed. The major issue is that due to how programs are written, this error could be a noticeable amount of time. For example, suppose that the function G took a multiple of a million cycles to run. As embedded processors operate in the MHz frequency range, this could amount to delays of a noticeable fraction of a second.

3. EVT assumes the IID property

EVT also makes the assumption that the data to be modeled is i.i.d., or independent and identically distributed [9]. As an example, consider the original purpose of the Gumbel distribution: modeling floodwater levels - a safety critical situation, where reliable results are needed. Floodwater data is normally independent; provided the landscape is unchanged and flood defences repaired but not enhanced, one flood does not affect another. If this is not the case, then future floods are said to depend on the flood which caused this change. Similarly, these future floods will also have a different distribution to the previous floods, due to the changes effecting how much water is needed to cause flooding. Hence, these future floods do not follow the same distribution as the past floods, and so are not identically distributed.

In statistics, the i.i.d. assumption simplifies things immensely, if it can be made. Unfortunately, the execution times of programs need not be independent or identically distributed.

An obvious example of runtimes being dependent is that of amortised data structures. In this case the runtime of adding an element to the data structure depends on the current state of the data structure. Avoiding amortised data structures is not enough though; depending on testing methodology, almost

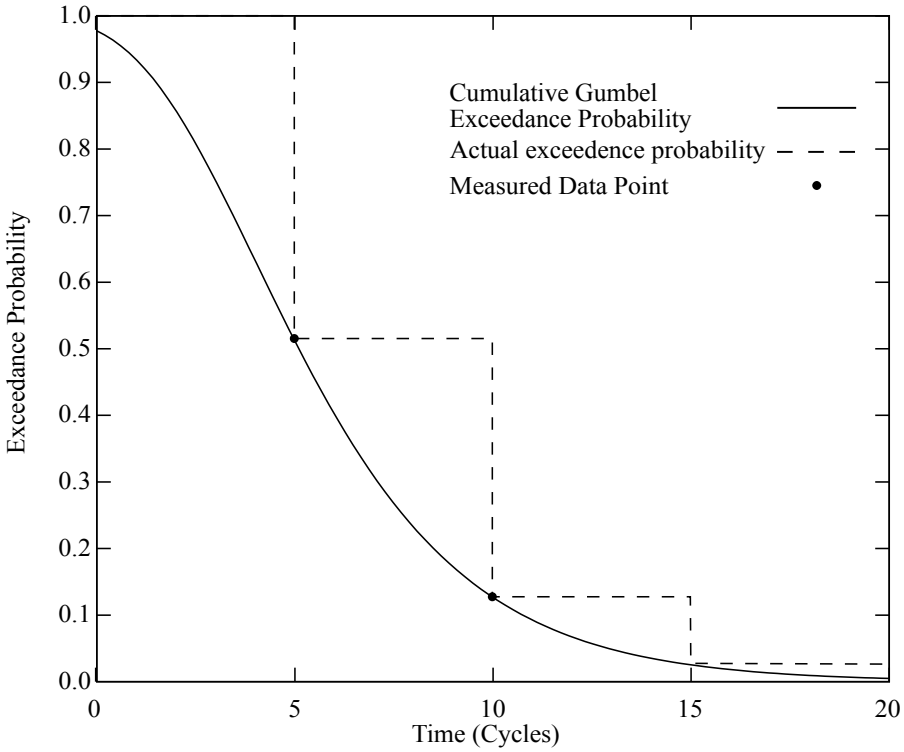

Figure 2. An illustration of how the Gumbel distribution can infer unsafe values due to the implicit assumption that a program may stop at any time during execution.

any program can have dependent runtimes, simply through an on-chip cache. If the program can leave the cache in multiple states depending on its input, then each runtime is dependent on what was run before it - and such a dependence may not be trivial to model. For instance, a long runtime may mean that more parts of the instruction code have been loaded into cache, decreasing the next runtime. Alternatively, it may mean that this run of the program performed a lot of cache thrashing - and hence leaving the cache in a less useful state for the next run of the program.

Further, some systems may necessarily have dependent runtimes, especially in the case of systems which manipulate a value in the real world. As an example, take an aircraft flight control system. As an input, the control system will consider the present velocity of the aircraft and produce an output which is a modification to that velocity. Hence the input, and therefore runtime, of any run of the system will depend on the outputs of every previous run.

A similar argument can be used to reason that program runtimes need not be identically distributed, in that each run of the program has the potential to "change the world". However, it turns out that the problem of runtimes not being identically distributed is more widespread. Suppose $X(n)$ is the uniform random distribution over the closed interval $[1, n]$, and that programs A, B produce different runtime distributions when run with data sampled from $X(100)$. Then define program C as follows:

$$C(n) = \begin{cases} A(\lfloor n/2 \rfloor) & \text{if } n \text{ is even} \\ B(\lfloor n/2 \rfloor) & \text{if } n \text{ is odd} \end{cases}$$

Then if C has its runtimes sampled with data from $X(200)$, the runtimes will follow two distributions - one for those which have odd input data, and one for those with even input data. Hence the runtimes cannot be treated as being identically distributed at face value. It is possible to combine multiple distributions into one, but this can cause a loss of accuracy when fitting the Gumbel distribution. Further, it would be necessary to ensure that all possible distributions of runtimes were found. Given that the number of paths through a program is potentially large, and each path potentially leads to a different distribution, this would cause an unacceptable amount of testing to be necessary.

It should also be noted that this observation correlates with the results presented in Edgar's work [5]. For experiments on branch prediction there is a strong argument for the results being i.i.d.: that the effects of other unknown processes on the branch prediction experiment can be modeled by some i.i.d. random variable. In turn, this leads to highly accurate results. In more complicated experiments, such as bubble sort, there is no such argument as the amount of work the algorithm does is a non-trivial property of the input. Consequently the results of Gumbel prediction are much poorer in the bubble sort experiment [5].

In related work, Petters [13] also stated that the i.i.d. assumption needs to hold for EVT statistics to be valid. Petters' approach is to use statistical methods to gain a confidence value on predicted runtime of "measurement blocks" of code, and combine these to get an overall WCET. To ensure the i.i.d. assumption holds, Petters proposes to randomise any potentially unknown element of the processor state at the beginning of each measurement block. By using these measurement blocks, the problems outlined above are avoided as there are limited paths through the code, and within these blocks hazards are encountered with fixed probabilities determined by the initial randomisation. The potential downside to Petters' work is that it is assumed that the measurement blocks are independent. Whilst for Petters' work, on modelling processor features, it can be ensured that measurement blocks are independent, it may not be a practical assumption when modelling entire systems.

4. Additional Properties Needed to use EVT

Given the issues raised in the previous sections, it is apparent that to make use of EVT it is necessary to take measures so that it is known that the lack of realism in the model does not produce unsafe estimates.

There are two potential solutions available. The first is to prove that the problems either do not apply, or are bounded. The second is to adapt the use of statistical analysis so that it does not encounter any problems.

4.1. Proving Strategies and Bounding Undesirable Behaviour

One strategy to avoid problematic effects is to prove that they do not exist, or to prove a bound on them. Neither of these strategies can be automated, as mathematical proof is not automatable. Fortunately, there are some general techniques which can be used, which are outlined here. The techniques focus on producing some form of bound on undesirable properties.

To cope with the effects of approximating a discrete distribution with a continuous distribution, there is a simple method. This is to determine the maximum error this introduces and to add this error to any prediction. This seems as if it could be difficult to accomplish, as the maximum error is not known; however, there is a simple method available which does not require this information. Instead

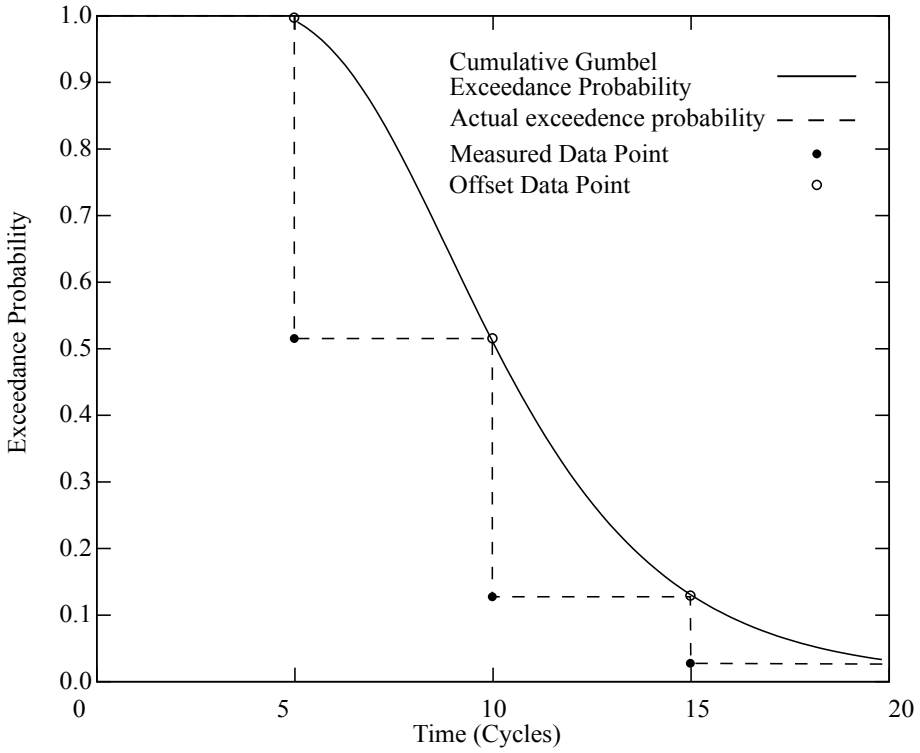

Figure 3. An illustration of how to offset the Gumbel distribution to guarantee safe values

of modeling the observed data directly, first convert the data to exceedance probabilities, such that value x_n is exceeded with probability p_n, for $n \in [1, k]$ and the x_n being original data points. Then model the data series of value x_n occurring with probability p_{n-1}, with $p_0 = 1$. This is illustrated in Figure 3 and moves the Gumbel estimate above the safety line by the minimum amount required to guarantee safety. The downside to moving the points in this way is that the estimates are pessimistic almost everywhere.

Another method to deal with errors from approximating a discrete distribution with a continuous distribution would be to try and argue that the effects are negligible. For instance, in the case when only a single path is being considered the error can be bounded by the largest single processor delay. This delay is likely to be measured in tens of processor cycles, and in if the processor speed is measured in MHz then the errors involved are likely to be negligible. Similar arguments could also be made if the different paths through the program have similar lengths and experience similar amounts of randomly-modeled hazards.

Proving the i.i.d. assumption is hard to do conclusively, with caches being an example of a common processor feature which causes the i.i.d. assumption to be violated. However, in some systems, particularly soft real time systems, it may be adequate to perform statistical analysis of the test data to determine if the i.i.d. assumption appears to hold. A statistical test for independence is as follows: Assume the null hypothesis that the execution times are independent of the system's previous actions. As the system's previous actions are determined by the system's input, it is sufficient to look for correlation between the runtime of a test and the inputs to previous tests. If correlation exists then the null hypothesis can be disproved; if correlation does not exist it only indicates that the test data does

not disprove the null hypothesis, and that over the entire set of test data the data appears independent to some degree of confidence. The test can be improved by splitting it into subtests of contiguous test data; this would prevent small runs of of dependent results from being hidden by a large quantity of independent results.

Another tactic to take into account dependence would be to bound it. This could easily be accomplished by a periodic reset of the system, which would break dependencies. Then the drift in execution times over this period could be found through a number of methods, including statistical analysis, as these are i.i.d. and very close to being continuous. For some systems, for instance the previously mentioned aircraft control system, a complete reset of the system may be impossible. In this case an appropriate strategy could be to periodically reset those resources which can be reset, and perform statistical analysis to gain some confidence that the remaining inputs to the system do not cause any dependencies between job executions.

Proving the identically distributed assumption is harder. The problem is that for every path through the program there will be a different set of hazards, leading to a potentially different distribution, and that it is difficult to argue that the worst case distribution is represented in the test data. If the worst case distribution is not represented, then statistical analysis cannot take it into account when combining all observed distributions into a Gumbel distribution. The simplest method here would be to ensure some level of code coverage, such as modified condition/decision coverage [1], and to ensure that each different path identified by the code coverage technique has sufficient samples taken from it to form a distribution.

4.2. Adapting Statistical Analysis

If it is not possible to bound the effects of not meeting the i.i.d. assumption, then the application of statistical analysis must be adapted instead. For this, it is necessary to look at what effects are being modeled, and identify what could cause problems. Given a sequential program and a system to run it on, it is reasonable to say that the execution time of the program depends on three properties:

- The input to the program.
- The initial state of any resources the program uses.
- Competition for resources from other programs (in preemptive or multiprocessor environments).

The third of these can be ignored, provided that testing is carried out in a representative environment including the programs which compete for the resource. However, it is necessary to enforce that programs do not directly communicate with each other, as this can introduce non-i.i.d. behavior. In such a case, it would be possible to use statistical analysis to predict the WCET of a group programs which do communicate heavily with each other, as then the dependencies become internal and invisible to the model.

Similarly, there is a simple solution to the second of these properties: perform a reset of the processor and shared resources between tests. Unfortunately, such an assumption may not be realistic or desirable in the real world, so an alternative would be to test multiple subsystems which all share the same resources at once. If such tests were randomly interleaved, then it becomes valid to treat the

initial state of the resources as another i.i.d. random variable, because for each test of a subsystem, any other test on any subsystem could have been carried out before, which effectively randomises the shared resources.

This only leaves the fact that input can effect runtime, leading to non-identically distributed runtimes, as a problem. The most obvious solution to this would be to fix the input such that it forces the worst case path through the program. The fixed input would guarantee that the measurements are sampled from the same distribution. The major concern with this approach is that due to low level processor details, such as floating point division, it is conceivable that using a single input will not be able to find the WCET. Hence it makes sense to widen the values used as input, when sampling, to the set of all inputs which lead to the worst case path through the program.

The biggest issue with the above remedy is that it requires knowledge of the worst case path through the program, but the worst case path through the program is not known - or WCET analysis wouldn't be necessary. However, other analysis methods may be able to determine a set of candidate paths through the program, of which one is the worst case path. This can be seen in abstract interpretation [7], which uses abstract states consisting of many concrete states to determine the worst case path. Statistical analysis could be used in this case to analyse each path in such a set individually to determine which path is the worst case, essentially reversing the abstraction. It may be that statistical analysis would indicate that all of the paths in the worst case abstract state present a better case than those in some other set, in which case the paths to be explored by abstract interpretation would have to be extended to this other set.

An alternative method of making the identically distributed assumption hold would be to use external measurements, such as the number of instructions executed and the number of cache misses, to create a catergorisation scheme. Provided that the chosen measurements lead to categories whose members are sampled from the same or similar distributions, the identically distributed assumption should hold. Then it becomes possible to work out a WCET time by applying statistical analysis to each category and picking the maximum. Alternatively, the categories could form the basis for using parametric WCET analysis techniques [11] to further enhance the precision of the results.

If the i.i.d. assumption is satisfied by using some of the methods outlined above, then it only remains to determine that the errors associated with fitting a smooth curve to the distribution are small and that the safety of the model is preserved. Fortunately, this turns out to be relatively straightforward: given the restrictions to make the measurements meet the i.i.d. requirement, the measurements should be similar enough that a significant error, of the type seen in Figure 2, cannot exist.

5. Conclusion

The use of statistical estimation for WCET is a powerful method, and can be used to model WCET with great accuracy [5, 6, 8]. However, as statistical estimation sacrifices realism it is necessary to take measures so that the safety of the model is not an unintended casualty of this sacrifice. To ensure the safety of the model it is either necessary to prove that the assumptions of EVT statistics hold or to adapt the application of statistical analysis. Adaption requires an additional set of restrictions on the measurements used to generate the statistical model, but in turn these should produce a more sound model.

References

[1] BADGETT, T., THOMAS, T. M., AND SANDLER, C. *The Art of Software Testing*, second ed. John Wiley and Sons, Inc, 2004.

[2] BERNAT, G., COLIN, A., AND PETTERS, S. WCET analysis of probabilistic hard real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE* (2002), pp. 279–288.

[3] BULLOCK, S., AND SILVERMAN, E. Levins and the legitimacy of artificial worlds. 2008.

[4] BUTLER, R. W., AND FINELLI, G. B. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Trans. Softw. Eng. 19*, 1 (1993), 3–12.

[5] EDGAR, S. *Estimation of Worst-Case Execution Time Using Statistical Analysis*. PhD thesis, University of York, 2002.

[6] EDGAR, S., AND BURNS, A. Statistical analysis of WCET for scheduling. In *Proceedings IEEE RTSS 2001* (2001).

[7] FERDINAND, C., HECKMANN, R., LANGENBACH, M., MARTIN, F., SCHMIDT, M., THEILING, H., THESING, S., AND WILHELM, R. Reliable and precise WCET determination for a real-life processor. In *EMSOFT* (2001), T. A. Henzinger and C. M. Kirsch, Eds., vol. 2211 of *Lecture Notes in Computer Science*, Springer, pp. 469–485.

[8] HANSEN, J., HISSAM, S. A., AND MORENO, G. A. Statistical-based WCET estimation and validation. In *9th Intl. Workshop on Worst-Case Execution Time Analysis, Dublin, Ireland* (2009), N. Holsti, Ed., vol. 252, Austrian Computer Society.

[9] KOTZ, S., AND NADARAJAH, S. *Extreme Value Distributions: Theory and Applications*. Imperial College Press, 2000.

[10] LEVINS, R. The strategy of model building in population biology. *American Scientist 54*, 4 (1966), 421–431.

[11] LISPER, B. Fully automatic, parametric worst-case execution time analysis. In *WCET* (2003), J. Gustafsson, Ed., vol. MDH-MRTC-116/2003-1-SE, Department of Computer Science and Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden, pp. 99–102.

[12] ODENBAUGH, J. The strategy of the strategy of model building in population biology. *Biology and Philosophy 5* (Nov 2006), 607–621.

[13] PETTERS, S. M. *Worst Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute for Real-Time Computer Systems, Technische Universit Mnchen, 2002.

[14] SEN, R., AND SRIKANT, Y. N. WCET estimation for executables in the presence of data caches. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software* (New York, NY, USA, 2007), ACM, pp. 203–212.