Optimal Restart Times for Moments of Completion Time

Aad P. A. van Moorsel
aadvanmoorsel.com
Goßlerstraße 11, 12161 Berlin,
Germany
conferences@aadvanmoorsel.com

and

Katinka Wolter
Humboldt-Universität Berlin
Institut für Informatik
Unter den Linden 6, 10099 Berlin,
Germany
wolter@informatik.hu-berlin.de

Abstract

Restart is an application-level technique that speeds up completion for jobs with highly variable completion times. In this paper, we present an efficient iterative algorithm to determine the optimal restart strategy if there are a finite number of restarts, and if one is interested in minimising higher moments of the completion time. We demonstrate its computational efficiency in comparison with alternative algorithms. We also discuss fast approximations to determine close to optimal restart times for limiting cases.

1 Introduction

The work in this paper applies to various forms of 'restart,' which simply implies that a task is retried after some time threshold has passed. Restart finds its application in areas ranging from randomised algorithms [2] to distributed data base queries [6], Internet agents [3] and software rejuvenation [1]. Probably the most commonly experienced restart mechanism is clicking the reload button of the web browser when a download takes too long.

The basic form of restart can be conveniently modelled assuming that (1) successive downloads are statistically independent and identically distributed, and (2) new tries abort previous tries. Such model assumptions have been found realistic for Internet applications [3, 5]. Under these assumptions, we provide in this paper an efficient algorithm to determine the optimal time instances at which to initiate restarts, so that higher moments of completion time are minimised.

As we showed in [4], the optimal restart strategy for *unbounded* number of restarts can be determined in straightforward manner, as can that for a finite number of restarts and the *first* moment. However, the combination of higher moments and finite number of restarts is considerably more challenging, and requires an iterative approach to deal with the multiple dimensions of the optimisation problem. Our proposed algorithm leverages various expressions for the moments of completion time to simplify the minimisation problem. Because of

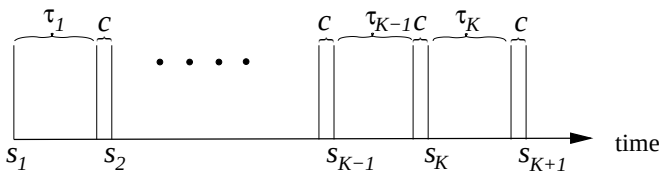

Figure 1: Labelling restarts, total of K restarts.

these simplifications, the algorithm outperforms more naive approaches by up to an order of magnitude, as we will show.

The paper also provides insights into the characteristics of the optimal restart strategy through approximations under limiting conditions. These approximations enable quick estimates for optimal restart times, and also show that, contrary to the situation for the first moment, it is typically not optimal to apply restarts at constant intervals when minimising higher moments.

2 Model

Let the random variable T represent the completion time of a job without restarts, $f(t)$ its probability density function, and $F(t)$ its distribution. The total number of restarts is K, and the overhead associated with restarting is c time units for each restart. The random variable T_K represents the completion time with K restarts, and the restart intervals have length $\tau_1, \ldots, \tau_K$, as shown in Figure 1. We also use $\tau_{k|K}$ (instead of τ_k) to denote the k-th restart time out of a total of K restarts. The k-th interval starts at time s_k. So, we get $s_1 = 0$, $s_2 = \tau_1 + c$, $s_3 = \tau_1 + c + \tau_2 + c$, etc., until $s_{K+1} = \sum_{k=1}^{K} \tau_k + Kc$.

Setting $\tau_{K+1} = \infty$ for notational purposes, we obtain density function f_K and survival function $\bar{F}_K = 1 - F_K$ piece-wise for each restart interval [4]:

$$\bar{F}_K(t) = \begin{cases} \prod_{i=1}^{k-1} \bar{F}(\tau_i)\bar{F}(t - s_k) & \text{if } \; s_k \le t < s_k + \tau_k, \quad k = 1, \ldots, K+1 \\ \prod_{i=1}^{k-1} \bar{F}(\tau_i) & \text{if } \; s_k + \tau_k \le t < s_{k+1}, \quad k = 1, \ldots, K \end{cases}$$

$$f_K(t) = \begin{cases} \prod_{i=1}^{k-1} \bar{F}(\tau_i) f(t - s_k) & \text{if } \; s_k \le t < s_k + \tau_k, \quad k = 1, \ldots, K+1 \\ 0 & \text{if } \; s_k + \tau_k \le t < s_{k+1}, \quad k = 1, \ldots, K \end{cases} \tag{1}$$

It is worth visualising the density of T_K, see Figure 2 for a lognormal distributed T, with $K = 3$ restarts, restart intervals $\{\tau_1, \tau_2, \tau_3\} = \{0.25, 0.2, 0.15\}$, and $c = 0.02$.

The N-th moment $E[T_K^N]$, our metric of interest, is by definition:

$$E[T_K^N] = \int_0^\infty t^N f_K(t) dt = \sum_{k=1}^{K+1} \int_{s_k}^{s_k + \tau_{k|K}} t^N f_K(t) dt. \tag{2}$$

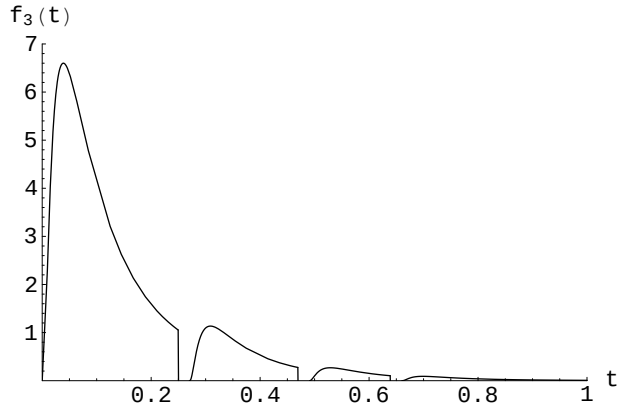

Figure 2: The probability density f_3 of completion time T_3 with restarts $\{\tau_1, \tau_2, \tau_3\} = \{0.25, 0.2, 0.15\}$, and $c = 0.02$ (based on lognormal distributed completion time T).

3 Optimisation

To find the restart times $\tau_{1|K}, \ldots, \tau_{K|K}$ that minimise $E[T_K^N]$, one could minimise (2) directly. It results in a K-dimensional minimisation problem that can be solved with off-the-shelf optimisation software. However, it is computationally expensive, since every new 'guess' for $\tau_{k|K}$ implies recomputing the integral term in (2) for all intervals $[s_l, s_l + \tau_{l|K})$ with $l \geq k$, to determine if the guess improves $E[T_K^N]$.

As an alternative, we can exploit an expression for $E[T_K^N]$ we derived in [4]. It recursively relates moments for K restarts with that for $K-1$ restarts by adding one restart before the existing $K-1$:

$$E[T_K^N] = M_K[T^N] + \bar{F}(\tau_{1|K}) \sum_{n=0}^{N} \binom{N}{n} (\tau_{1|K} + c)^{N-n} E[T_{K-1}^n], \tag{3}$$

where $M_K[T^N]$ denotes the 'partial moment,' defined for $k = 1, \ldots, K$, as:

$$M_k[T^N] = \int_0^{\tau_{1|k}} t^N f(t) dt.$$

Hence, instead of minimising (2) one can minimise (3). In this case, however, every new 'guess' for τ_k implies computing $E[T_K^N]$ 'all the way,' recursively computing $E[T_l^N]$ for all values $l \geq k$, to determine if the guess improves $E[T_K^N]$. This also introduces much computational overhead. (In Figure 5 we will see that minimising (3) is in fact slightly less expensive than minimising (2), at least for the discussed example.)

Instead of the direct minimisation of (2) or (3), we would like to extend to higher moments an idea that worked very well in [4] for the first moment. Utilising the recursion in (3), we derived an algorithm in [4] that sequentially determines the restart time $\tau_{1|k}$ that minimises $E[T_k] = M_k[T] + \bar{F}(\tau_{1|k})(\tau_{1|k} + c + E[T_{k-1}])$, for $k = 1$ until K. Its correctness relies on the

fact that the optimal restart time $\tau_{K-k+1|K}$ equals $\tau_{1|k}$, because the first moment is insensitive to shifts [4]. We named this algorithm the backward algorithm, since it traverses backward in time. A single pass of K optimisations is guaranteed to provide the optimal restart times. The backward algorithm is thus much more efficient than minimising (2) or (3). (Figure 5 shows an improvement of about a factor 20.)

The problem in applying this idea to higher moments is that the restart time $\tau_{1|k}$ that minimises $E[T_k^N]$ in general is not equal to its counterpart $\tau_{K-k+1|K}$ that minimises $E[T_K^N]$. Higher moments are not insensitive to shifts. To resolve this issue, we need the following theorem.

Theorem 1. *For any $k, k = 1, \ldots, K$, the 'last' k restart times $\tau_{K-k+i|K}$, $i = 1, \ldots, k$, minimise $E[T_K^N]$ (given other restart times $\tau_{l|K}$, $l = 1, \ldots, K-k$) if and only if $\tau_{i|k} = \tau_{K-k+i|K}$, where $\tau_{i|k}$ minimises $E[(T_k + s_{K-k+1})^N]$.*

Proof. First, by definition:

$$E[T_K^N] = \int_0^{s_{K-k+1}} t^N f_K(t)dt + \int_{s_{K-k+1}}^{\infty} t^N f_K(t)dt.$$

Since the left most integral term does not depend on $\tau_{K-k+i|K}$, $i = 1, \ldots, k$, the last k optimal restart times minimise $E[T_K^N]$ if and only if they minimise $\int_{s_{K-k+1}}^{\infty} t^N f_K(t)dt$. Then, it follows from (1) that for every set $\tau_{K-k+i|K}$, $i = 1, \ldots, k$, there exists a corresponding set $\tau_{i|k}$, with $\tau_{i|k} = \tau_{K-k+i|K}$, $i = 1, \ldots, k$, and such that for any $t \geq 0$:

$$f_K(t + s_{K-k+1}) = \prod_{l=1}^{K-k} \bar{F}(\tau_{l|K}) f_k(t).$$

This implies that:

$$\begin{aligned} \int_{s_{K-k+1}}^{\infty} t^N f_K(t)dt &= \int_0^{\infty} (t + s_{K-k+1})^N f_K(t + s_{K-k+1})dt \\ = \prod_{l=1}^{K-k} \bar{F}(\tau_{l|K}) \int_0^{\infty} (t + s_{K-k+1})^N f_k(t)dt &= \prod_{l=1}^{K-k} \bar{F}(\tau_{l|K}) E[(T_k + s_{K-k+1})^N]. \end{aligned}$$

The product in this expression is independent of $\tau_{K-k+i|K}$, $i = 1, \ldots, k$, and thus minimising $\int_{s_{K-k+1}}^{\infty} t^N f_K(t)dt$ corresponds to minimising $E[(T_k + s_{K-k+1})^N]$. Combining the above arguments, we see that if $\tau_{K-k+i|K}$, $i = 1, \ldots, k$, minimises $E[T_K^N]$, then the corresponding restart times $\tau_{i|k}$, $i = 1, \ldots, k$, minimise $E[(T_k + s_{K-k+1})^N]$. □

Theorem 1 implies that for any k, $k = 1, \ldots, K$, determining the optimal restart time $\tau_{K-k+1|K}$ is equivalent to finding the restart time $\tau_{1|k}$ that minimises:

$$E[(T_k + s_{K-k+1})^N] = \sum_{n=0}^{N} \binom{N}{n} s_{K-k+1}^{N-n} E[T_k^n], \tag{4}$$

where $E[T_k^n]$ obeys (3): $E[T_k^n] = M_k[T^n] + \bar{F}(\tau_{1|k}) \sum_{m=0}^{n} \binom{n}{m} (\tau_{1|k} + c)^{n-m} E[T_{k-1}^m]$. Figure 5 in Section 6 shows that for our example it saves about 70 percent computer time to optimise the restart times in this way. The benefit of using (4) is that with every 'guess' in the optimisation algorithm it neither requires to recompute integral terms for all values $l \geq k$ (as in (2)), nor terms $E[T_l^N]$ for all values $l \geq k$ (as in (3)). One can say that we have isolated the optimisation of the k-th restart time from its interference with the other restart intervals.

4 Algorithm

The resulting optimisation algorithm is given as Algorithm 1. Contrary to the backward algorithm for the first moment [4] it does not terminate in K steps, but instead requires to iterate until either $E[T_K^N]$ or the restart times converge. One can apply generic approaches to decide which restart time τ_k to optimise at each iteration (such as the method of steepest descent). However, we propose three particular ways, which try to leverage the structure of the problem to minimise computation: backward, forward and alternating.

Algorithm 1 (Backward, Forward and Alternating Optimisation).

```
Choose constants N and K;
For n = 1 to N
    Compute E[T_0^n];
Determine τ_∞ that minimises E[T_∞];
For k = 1 to K
    Initialise τ_k = τ_∞;
While( not converged ) Do {
    For k = K to 1 {
        If ( backward || alternating ) then
           Set τ_k so that it minimises τ_{1|k} in (4), using (3) for E[T_k^n];
        For n = 1 to N
           Compute E[T_k^n] using (3);
    }
    If( forward || alternating ) then {
        For k = 1 to K
           Set τ_k so that it minimises τ_{1|k} in (4), using (3) for E[T_k^n];
    }
}
Return τ_1, ..., τ_K;
```

Note that every minimisation step can be carried out with a general-purpose optimisation algorithm. Also, note that $E[T_\infty]$ can be minimised using the expression $E[T_\infty] = (M_\infty[T] + \bar{F}(\tau_\infty)(\tau_\infty + c))/F(\tau_\infty)$ derived in [4]. The reason to initialise the algorithm with τ_∞ will become apparent when discussing the bulk approximation in the next section.

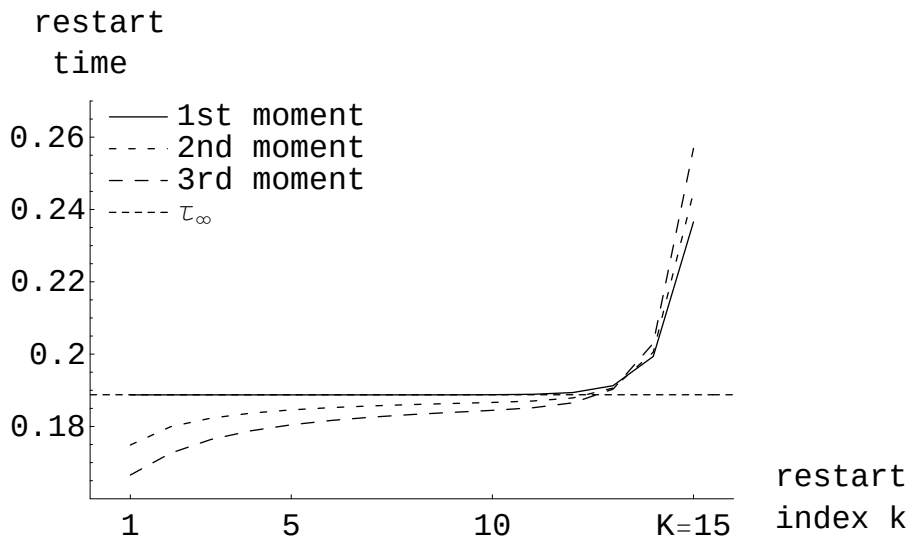

Figure 3: Optimal restart times, with respect to the moments $E[T_{15}]$, $E[T_{15}^2]$ and $E[T_{15}^3]$, respectively.

5 Characteristics of Optimal Restart Times

We applied our algorithm to a lognormal distribution, with parameters $\mu = -2.31$ and $\sigma = 0.97$.[1] We determine $K = 15$ restart times that minimise the first, second and third moment of the completion time. These restart times (with an interpolating curve) are shown in Figure 3. The figure also shows τ_∞, which is the starting solution set at the initialisation step in Algorithm 1.

Figure 3 indicates that when minimising the first moment, the optimal restart time $\tau_{k|K}$ monotonically converges when k gets smaller, to a single optimum τ_∞, as long as K is large enough. For higher moments, things are not as straightforward, as also witnessed by Figure 3. Nevertheless, convergence does get established, albeit with more intricate patterns than for the first moment. To show this, we study three limiting cases, namely at the right boundary ($\tau_{k|K}$ for $k \to K$, and $K \to \infty$), middle or 'bulk' ($\tau_{k|K}$ for $k \to \infty$ and $K - k \to \infty$), and left boundary ($\tau_{k|K}$ for $k \downarrow 1$ and $K \to \infty$). Figure 4 illustrates the main results. Space limitations do not allow us to go into all detail, but we introduce all three limiting cases briefly.

Right boundary approximation. For $K \to \infty$ (and non-diminishing optimal restart times) it follows that $s_K \to \infty$. As a consequence, optimisation of τ_K using (4) is dominated by the term $s_K^{N-1} E[T_1]$. Hence, for *any* moment N, the best restart time at the right boundary when $K \to \infty$ tends to the optimal restart time for the *first* moment $E[T_1]$. This optimum can be determined by finding the restart time $\tau_{1|1}$ that minimises $E[T_1] = M_1[T] + \bar{F}(\tau_{1|1})(\tau_{1|1} +$

[1]For the current paper, there is no particular significance to the chosen parameter values. They happen to be the parameters of a lognormal fit for experimental data of HTTP GET completion times [5].

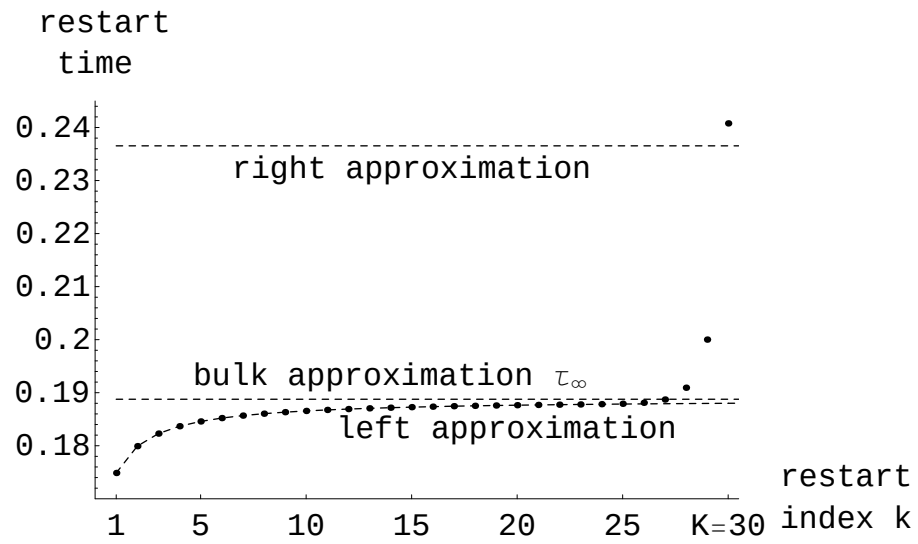

Figure 4: The dots give restart times that minimise the second moment $E[T_{30}^2]$, the dashed lines are the approximations, as labelled.

$c + E[T]$) [4]. Figure 4 shows the right approximation and it turns out that for $K = 30$ it is reasonable but not exceptionally close to the actual optimal restart (given by the dot).

Bulk approximation τ_∞. At the 'bulk,' or the middle of the pack, we get a limiting result if k is both far away from 1 and from K. We write this as $k \to \infty$, and $K - k \to \infty$. In that case, it follows in the same way as for the right boundary that the minimum is dominated by the first moment, which converges to τ_∞ (as discussed above, and shown in Figure 3). For higher moments, restart times thus tend to τ_∞ at the bulk. This also explains why we chose τ_∞ as initial solution in Algorithm 1: it is close to optimal for the bulk of restarts. In Figure 4, τ_∞ is indeed close to the optimal restart times, although not as close as the following approximation at the left boundary.

Left boundary approximation. At the left boundary, we can get a limiting result for restart intervals close to time 0. That is, we get an approximation for $\tau_{k|K}$ with $k \downarrow 1$ and $K \to \infty$. It is obtained by applying Algorithm 1 to the distribution of the completion time with restarts every τ_∞ time units, building forth on the limiting result at the bulk, and using expressions derived in [4]. Here the approximation is remarkably close, as seen from Figure 4. In fact, other experiments indicate that the left boundary approximation is very close irrespective of the value of K. Perhaps one would have expected the optimal restart times to turn quicker towards τ_∞ as k increases, but they do not move away from the left boundary approximation until a few restarts from the end (about $k = K - 4$).

As a conclusion, we see from the approximations that τ_∞ is a pretty good restart time, independent of the moment one is interested in. We have also seen that for the first moment, if an unbounded number of restarts takes place, a restart strategy with a single restart time τ_∞ is optimal (the precise conditions under which this is true need to be investigated). The

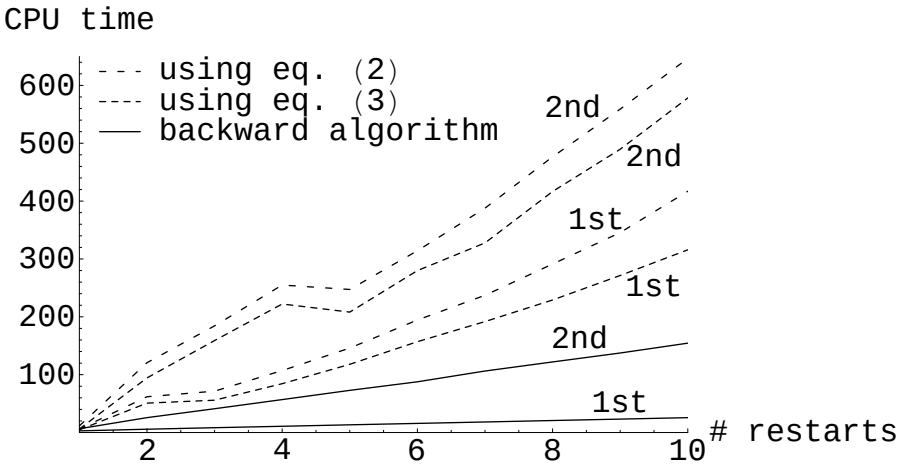

Figure 5: CPU time used for different algorithms, applied to minimise first as well as second moment.

convergence behaviour for the higher moments, however, indicates that if we allow an unbounded number of restarts, a restart strategy with a single value for all restart times is *not* optimal. The left boundary behaviour disrupts this.

6 Computational Effort

In Figure 5 we plot the time used for three different methods: Algorithm 1 (backward), minimising expression (2), and minimising expression (3). In all cases we applied default minimisation algorithms in Mathematica to carry out the optimisation steps. Algorithm 1 outperforms the other approaches. For the first moment, the speed up is an order of magnitude, which finds its explanation in assured convergence in K steps of the backward algorithm. For the higher moments, the speed up is about a factor 3 or 4. Apparently, the arguments put forward in Section 3 hold correct. We note that we tuned our Mathematica program to the best of our abilities, memorising in-between results so that the recursion in (3) and repetitive computation in (2) do not become bottlenecks. We also set the convergence criteria similarly, being determined by convergence of $E[T_K^N]$. Hence, although we do not have access to the 'internals' of Mathematica's optimisation algorithm, we are reasonably confident that the comparison of the three approaches is fair.

Figure 6 compares the three versions of Algorithm 1: backward, forward and alternating. These three exhibit similar performance. For our example, the forward algorithm turns out to require one pass less through all restart times than the other two algorithms, and hence it takes less CPU time.[2] Typically, we require not more than 5 passes through the K restart times,

[2]Note that because of the workings of the Mathematica optimisation algorithm, the comparison in Figure

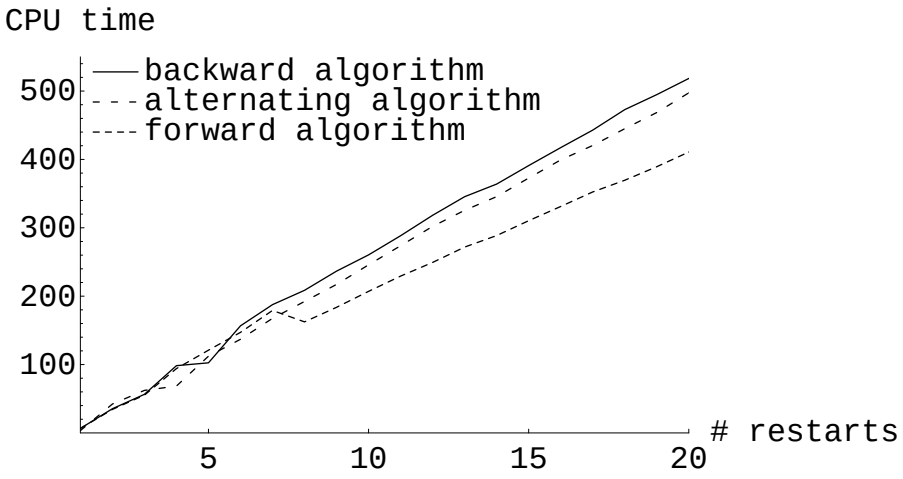

Figure 6: CPU time used by three versions of algorithm.

irrespective of the value of K. Studying the complexity of our Mathematica implementation, it turns out that running the optimisation routine is the computationally most expensive part: at step k, optimisation of τ_k takes an order of magnitude more time than the computation that updates $E[T_k^N]$. Algorithm 1 uses backward and/or forward traversal through the K restart times for reasons of computational efficiency. The algorithm may therefore be further improved by choosing the order in which to optimise restart times based on criteria such as steepest descent, which may decrease the amount of calls to the optimisation routine. This requires more experimentation.

References

[1] A. Bobbio, S. Garg, M. Gribaudo, A. Horvath, M. Sereno and M. Telek, "Modeling Software Systems with Rejuvenation, Restoration and Checkpointing through Fluid Stochastic Petri Nets," in *Proceedings of PNPM '99,* Zaragoza, Spain, pp. 82-91, IEEE CS Press, Sept 1999.

[2] M. Luby, A. Sinclair and D. Zuckerman, "Optimal Speedup of Las Vegas Algorithms," *Israel Symposium on Theory of Computing Systems,* pp. 128–133, 1993.

[3] S. M. Maurer and B. A. Huberman, "Restart strategies and Internet congestion," in *Journal of Economic Dynamics and Control,* vol. 25, pp. 641–654, 2001.

[4] A. van Moorsel and K. Wolter, "Analysis and Algorithms for Restart," *submitted for publication*, Apr. 2004.

5 is based on convergence of $E[T_K^N]$ as stopping criterion, while that in Figure 6 is based on convergence of restart times, a stricter criterion. This explains the higher CPU usage for the backward algorithm in Figure 6 compared to Figure 5.

[5] P. Reinecke, A. van Moorsel and K. Wolter, "A Measurement Study of the Interplay between Application Level Restart and Transport Protocol," *Service Availability Forum*, Munich, May 2004.

[6] Y. Ruan, E. Horvitz and H. Kautz, "Restart Policies with Dependence among Runs: A Dynamic Programming Approach," in *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming,* Ithaca, NY, Sept. 2002.