To: MLS Group
Subject: Hashing
From: Van Snyder

During a presentation about how MLS programs processes configuration files, a question was asked "what is this hashing stuff?" This description is from a memo that Fred Krogh and I wrote in 1983.

A Re-examination of Hashing Using Separate chains

Fred T. Krogh
W. Van Snyder

Section 366
Computing Memorandum No. 496
January 19, 1983
Revised February 4, 1983
Revised November 22, 1983

Abstract

A discussion and analysis of a well known but frequently-discounted hashing algorithm is presented. We believe its use is preferred in most hashing applications.

CR Categories and Subject Descriptors: D.2.8 [Software Engineering]" Metrics – *performance measures*; E.2 [Data]: Data Storage Representations – *Hash-table representations*; F.2.2 [Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems – *sorting and searching*; H.2.2 [Database management]: Physical Design – *access methods*; H.3.3 [Information storage and retrieval]: Information Search and Retrieval – *search process*

General terms: Algorithms, Design, Performance, Theory

Additional Key Words and Phrases: analysis of algorithms, hashing, data structures

Introduction

The fundamental idea of hashing is that the expected location of an entry in a table, called the initial probe location, or hash value, is computed using some part, called the key, of the entry. An algorithm for inserting entries into a hash table must cope with the possibility that entries having different keys might result in the same initial probe location. When such a collision occurs, only one of the entries can be stored at the initial probe location; the others must be stored somewhere else. The principal difference between hashing algorithms is the way they cope with collisions.

In [3] Vitter describes algorithms for hashing that use coalesced chains and a cellar. When a collision occurs, and the entry at the initial probe location has a key different from the key of the new entry, a chain is searched from the initial probe location to determine if the new entry is in the table. Each examination of an entry in the table is called a probe. If the new entry is not in the chain, it is entered in an empty location, and the chain is extended to point to it. The chain is said to be coalesced because entries having different initial probe locations might be members

of the same chain. The cellar is an area of the hash table used only to store members of chains. Initial probe locations are never in the cellar.

It is the purpose of this note to call attention to an algorithm that does not allow chains to coalesce. That is, every entry in a chain has the same hash value, and entries in different chains have different hash values. In addition, one may assume that the first member of every chain is stored at the location P indicated by the hash value. As a consequence of this construction, a non-empty entry at location J either has a hash value equal to J, or it is a member of a chain of entries having a hash value different from J, and there is no entry having a hash value equal to J.

When given an entry G to be inserted, an algorithm to construct a hash table using separate chains must consider three conditions of the entry A at P, the hash value of G:

1. A is an empty entry, in which case G may be stored at P;
2. the hash value of A is equal to P, in which case G may be stored in any empty location, but must be made a member of the chain that starts at P;
3. the hash value of A is different from P, in which case the entry at P must be moved to any empty location, resulting in the first condition above.

Since chains do not coalesce, the expected number of probes necessary to locate an entry is smaller, a cellar is almost pointless because the use of a cellar does not decrease the expected number or probes, deletion of entries is simpler, and in some implementations space for links that define chains can be found in the space ordinarily used for the key of an entry. Inserting new entries is slightly more complicated, but in our view the complication is justified.

Notation

$M =$ Size of the table.
$N =$ Number of non-empty entries.
$\alpha = \frac{N}{M}$.
$A =$ An entry in the table.
$k(A) =$ Key of A.
$h(A) \equiv h(k(A)) =$ Initial probe location.
A_i, $1 \le i \le k =$ Chain of entries such that $h(A_i) = h(A_1)$. A chain is ordered if for some function $f(A)$ the condition $f(A_i) < f(A_{i+1})$, $1 \le i \le k$ is imposed.
$C_i =$ Location of A_i, the i^{th} entry in a chain.
L_i, $1 \le i \le k =$ Link field contained in $A_i \equiv C_{i+1}$.
$L_k = C_1$ or an end mark.
$\mu =$ (cost of moving an entry) / (cost of probe).
$\sigma =$ (number of successful searches) / (number of entries added to the table).
$\nu =$ (number of unsuccessful searches) / (number of entries added to the table).
$\phi =$ (cost of following a link) / (cost of a probe) $\equiv 0.5$.
$S(\alpha) =$ Expected probes for successful search when using unordered chains.

$U(\alpha) =$ Expected probes for unsuccessful search when using unordered chains.

$V(\alpha) =$ Probability that an entry must be moved when using unordered chains.

$L(\alpha) =$ Expected number of links that must be extracted from entries in a chain to find the predecessor of any entry that must be moved when using unordered chains.

$S', U', V', L' =$ Quantities of S, U, V, L when using ordered chains.

S_V, $U_V =$ S and V for Vitter's algorithm C with $\beta = 0.86$, see [3].

$E[S(\alpha)] =$ Expected value of $S(\alpha)$, $\frac{1}{\alpha}\int_0^\alpha S(t)\,\mathrm{d}t < E[S(\alpha)] < S(\alpha)$. $E[S(\alpha)]$ is usually near $S(\alpha)$.

$E[U(\alpha)] = \frac{1}{\alpha}\int_0^\alpha U(t)\,\mathrm{d}t$, with similar formulas for $E[V(\alpha)]$ and $E[L(\alpha)]$.

Algorithm S: "Look Up and Insert If Not Found"

S$_1$: Compute the initial probe location $P = h(G)$ for a given entry G. If the entry A at P is empty, store G at P and exit with an indication that G was inserted. If $k(A) = k(G)$, exit with an indication the G was found.

S$_2$: If $h(A) \neq P$ then go to step **S**$_4$. In most instances this may be determined efficiently using a bit in every (non-empty) entry.

S$_3$: $h(A) = P$. Follow the chain starting at P. If the key of an entry in the chain $= h(G)$, exit with an indication the G was found. Otherwise, find an empty entry, E, link the last location in the chain to E, set L_E to P or an end mark, and exit with in indication that G was inserted.

S$_4$: Search the chain of which A is a member to find the entry with $L_i = P$. This can be done either by hashing A and searching from $h(A)$, or by following a circular chain from P. The appropriate choice depends upon the relative costs of computing the initial probe and of following a linked list.

S$_5$: Find an empty entry E, move A to E, set L_i to E, insert G at P, and exit with in indication that G was inserted.

The procedure "find an empty entry, E" can be performed by remembering the location of empty space nearest the end of the table, by searching backward from E to find the next empty location, or by any other algorithm you like.

Deletion of an entry is very simple, and results in an "empty" entry, as opposed to the "deleted" entry required by some algorithms.

Algorithm **S** is easy to extend for use in a paged environment or for use on secondary storage.

Discussion

Knuth discusses this algorithm in [2, page 518], but prefers the algorithm obtained by deleting steps **S**$_2$, **S**$_4$, and **S**$_5$. Algorithm **S** does require the extra work to check whether an entry in the table is the start of a chain, to follow the chain to find the preceding entry, and to move an entry at $h(G)$ if it is not the start of a chain. But these actions are not needed often, and since the expected length of chains is small the total amount of work is small. Knuth shows the difference in the expected

number of probes in [2, fig. 44 on page 539]. The best choice clearly depends upon the expected number of lookups per entry, and the extra work mentioned above.

Table 1 gives formulas for the work performed by algorithm **S** for the cases when chains are ordered and unordered, and these results are quantified in Figure 1. The unordered case requires more probes but fewer moves than the ordered case. If $\mu \ll 1$ or $\nu \gg 1$, then the ordered algorithm is to be preferred. Since μ is rarely $\ll 1$, the unordered algorithm is essentially always preferred when $\nu \approx 1$.

In [3, pp 920-921] Vitter discounts algorithm **S** described above because of the cost of moves. (There is an algorithm **S** described in [3, p 918], but it is different from the algorithm **S** described above.) But entries must be unusually large (especially if the environment provides a block-move instruction), or there must be pointers to entries other than L_i, for moves to be of concern. Figure 2 quantifies the expected work for Vitter's algorithm C as compared to algorithm **S**. Let $g(\alpha;\mu,\nu,\sigma,\phi) = E[U_V(\alpha)] - E[U(\alpha)] - \frac{\phi}{\nu}E[L(\alpha)] - \frac{\mu}{\nu}E[V(\alpha)] + \sigma(E[S_V(\alpha)] - E[S(\alpha)])$. Given α, μ, ν, σ, and ϕ one would choose algorithm **S** if $g(\alpha;\mu,\nu,\sigma,\phi) > 0$ and Vitter's algorithm **C** otherwise. A good rule of thumb is that algorithm **S** will perform better than Vitter's algorithm **C** when $\mu < \sigma$ and vice versa for $\sigma < \mu$. For $\mu \approx \sigma$ algorithm **S** is relatively better for small or large α, and Vitter's algorithm **C** is better for $\alpha \approx 0.6$. We believe $\mu < \sigma$ to be the more common case.

The algorithm given by Brent in [1] is apparently the algorithm of choice if one is doing more than a few lookups per entry, there are no pointers to entries, and links are not practical. Knuth, however, mentions in [2] an idea due to Butler Lampson that in many cases allows one to create space for the links: Let $h(k(A)) = k(A) \mod M$. Thus $k(A)$ has the form $q * M + C_1$. Let $W = q * m + \text{link} = k(A) + \text{link} - C_1$. Then one can store W instead of $k(A)$, and reconstruct $k(A)$ from W when necessary. If one restricts the key to be positive and prohibits $W = \text{zero}$ for a non-empty entry (link = zero or q = zero), then $W < 0$ can indicate that the entry does not start a chain, and $W = 0$ can indicate an empty entry. By using algorithm **S** and Lampson's idea, one could construct a smaller algorithm that uses exactly the same space per entry and provides the same function as the Fortran program given in [1], subject to the restriction that the key must be positive (the Fortran program given in [1] does not allow the key to be -1 or zero), but with better performance. Knuth shows the difference in the expected number of probes in [2, fig. 44 on p 539].

When chains coalesce, Lampson's idea cannot be used, because C_1 cannot always be discovered and therefore $k(A)$ cannot always be computed. If the bits in an entry were used as efficiently as possible, Lampson's idea could be applied in the case of separate chains, and the case of coalesced chains would necessarily require that entries be sufficiently larger to store the link. If the same space were used in both cases, a double-linked chain could be constructed in the case of separate chains, thereby reducing the cost of inserting an entry because the predecessor of an entry that must be moved could be found immediately. Algorithm **S** and Vitter's Algorithm **C** could then be compared by using $g(\alpha;\mu,\nu,\sigma,\phi) = E[U_V(\alpha)] - E[U(\alpha)] - \frac{(\phi+\mu)}{\nu}E[V(\alpha)] + \sigma(E[S_V(\alpha)] - E[S(\alpha)])$ where ϕ in this case is the relative cost of following a backward link.

Table 1: Expected performance of Algorithm **S***

Expected Number of	Unordered chains Arbitrary α	Unordered chains $\alpha = 1$	Ordered chains Arbitrary α	Ordered chains $\alpha = 1$
Probes for successful search, $S(\alpha)$, $S'(\alpha)$	$1 + \frac{\alpha}{2}$	1.5	$1 + \frac{\alpha}{2}$	1.5
Probes for unsuccessful search, $U(\alpha)$, $U'(\alpha)$	$e^{-\alpha} + \alpha$	1.3679	$\frac{1}{2}(e^{-\alpha} + \alpha + 1)$	1.1839
Probes per entry to insert $N = \alpha M$ entries**, $E[U(\alpha)]$, $E[U'(\alpha)]$	$\frac{1}{\alpha}(1 - e^{-\alpha} + \frac{1}{2}\alpha^2)$	1.1321	$\frac{1}{2\alpha}(1 - e^{-\alpha} + \frac{1}{2}\alpha^2 + \alpha)$	1.0661
Moves to insert an entry, $V(\alpha)$, $V'(\alpha)$	$e^{-\alpha} + \alpha - 1$	0.3679	$\frac{1}{\alpha}(1 - e^{-\alpha}) + \alpha - 1$	0.6321
Moves per entry to insert $N = \alpha M$ entries**, $E[V(\alpha)]$, $E[V'(\alpha)]$	$\frac{1}{\alpha}(1 - e^{-\alpha} + \frac{1}{2}\alpha^2) - 1$	0.1321	***	0.2966
Links looked at for moves, $L(\alpha)$, $L'(\alpha)$	α^2	1	α^2	1
Links examined per entry for moves while inserting $N = \alpha M$ entries, $E[L(\alpha)]$, $E[L'(\alpha)]$	$\frac{1}{3}\alpha^2$	$\frac{1}{3}$	$\frac{1}{3}\alpha^2$	$\frac{1}{3}$

* We assume the initial probes are independent and uniformly distributed.

** If $\nu = 1$ this is also the expected number of probes per entry for unsuccessful searches.

*** This is $\frac{\alpha}{4} + \sum_{k=2}^{\infty} \frac{(-\alpha)^k}{(k+1)(k+1)!} = \frac{E_1(\alpha)+\ln(\alpha)+\gamma}{\alpha} + \frac{\alpha}{2} - 1$, where E_1 is the exponential integral and $\gamma \approx 0.57721\,56649$ is Euler's constant.

Figure 1: Comparison of ordered and unordered chains using Algorithm **S**

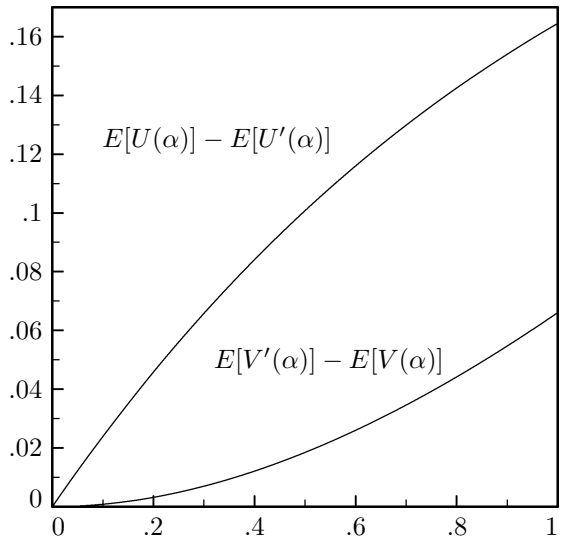


Figure 2: Comparison of Algorithm **S** and Vitter's Algorithm **C**

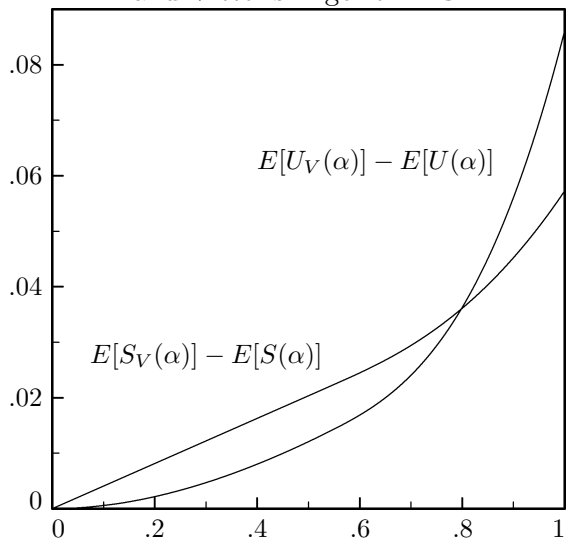
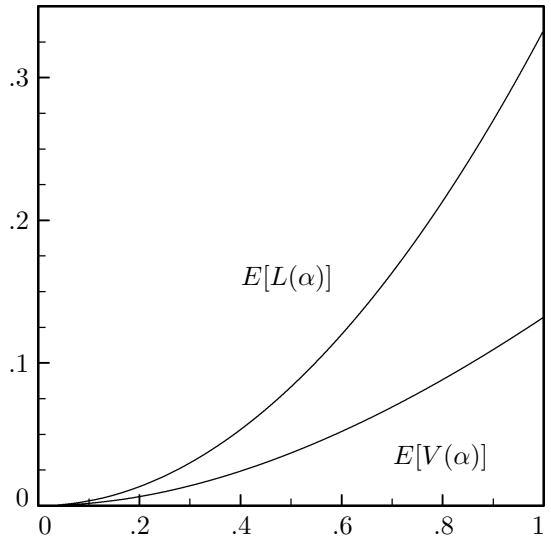


Figure 3: Moves, and links examined for moves, using Algorithm **S**



References

1. Richard P. Brent. Reducing the time of scatter storage techniques. *Communications of the ACM*, 16(2), February 1973.

2. Donald E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching.* Addison-Wesley, 1973.

3. Jeffrey S. Vitter. Implementations for coalesced hashing. *Communications of the ACM*, 25(12), December 1982.

Appendix
Analysis of work expected to be performed by Algorithm S

Figures 1 – 3 summarize the results obtained in these appendices.

Let $p_k(\alpha)$ be the probability that a given entry in the table *starts* a chain of length k or more when the table contains $N = \alpha M$ non-empty entries. Clearly $p_0(\alpha) = 1$ for all α. For a small increase δ in α, to terms of first order in δ

$$p_k(\alpha + \delta) - p_k(\alpha) = \delta(p_{k-1}(\alpha) - p_k(\alpha)). \tag{1}$$

This formula simply reflects the fact that the increase in chains of length k or more will come from chains of length of exactly $k - 1$. Dividing by δ and taking the limit as $\delta \to 0$,

$$\frac{\mathrm{d}p_k(\alpha)}{\mathrm{d}\alpha} = p_{k-1}(\alpha) - p_k(\alpha),\ p_0(\alpha) = 1,\ p_k(0) = 0 \text{ for } k > 0. \tag{2}$$

The solution of Equation (2) is

$$p_k(\alpha) = 1 - e^{-\alpha} \sum_{i=0}^{k-1} \frac{\alpha^i}{i!} = 1 - e^{-\alpha}\, e_{k-1}^{\alpha} = 1 - \frac{\Gamma(k,\alpha)}{(k-1)!},\ k > 0. \tag{3}$$

Clearly, the probability $q_k(\alpha)$ that a given entry starts a chain of length exactly k is given by

$$q_k(\alpha) = p_k(\alpha) - p_{k+1}(\alpha) = \frac{e^{-\alpha}\alpha^k}{k!} \tag{4}$$

A very brief derivation of the results in Table 1 follows.

$$S(\alpha) = S'(\alpha) = \frac{1}{\alpha} \sum_{k=1}^{\infty} \frac{k(k+1)}{2}\, q_k(\alpha) = e^{-\frac{\alpha}{2}} \sum_{k=0}^{\infty} (k+2) \frac{\alpha^k}{k!} = 1 + \frac{\alpha}{2} \tag{5}$$

$$E[S(\alpha)] = E[S'(\alpha)] = \frac{1}{\alpha} \int_0^{\alpha} S(a)\, \mathrm{d}a = 1 + \frac{\alpha}{4} \tag{6}$$

$$U(\alpha) = q_0(\alpha) + \sum_{k=1}^{\infty} k\, q_k(\alpha) = e^{-\alpha} + e^{-\alpha} \sum_{k=1}^{\infty} k\, \frac{\alpha^k}{k!} = e^{-\alpha} + \alpha \tag{7}$$

$$E[U(\alpha)] = \frac{1}{\alpha} \int_0^{\alpha} U(a)\, \mathrm{d}a = \frac{1}{\alpha}(1 - e^{-\alpha} + \frac{1}{2}\alpha^2) \tag{8}$$

$$U'(\alpha) = q_0(\alpha) + \sum_{k=1}^{\infty} \frac{k+1}{2}\, q_k(\alpha) = e^{-\alpha} + \frac{e^{-\alpha}}{2} \sum_{k=1}^{\infty} (k+1)\, \frac{\alpha^k}{k!} = \frac{e^{-\alpha} + \alpha + 1}{2} \tag{9}$$

$$E[U'(\alpha)] = \frac{1}{\alpha} \int_0^{\alpha} U'(a)\, \mathrm{d}a = \frac{1}{2\alpha}(1 - e^{-\alpha} + \frac{1}{2}\alpha^2 + \alpha) \tag{10}$$

$$V(\alpha) = \sum_{k=2}^{\infty} (k-1)\, q_k(\alpha) = e^{-\alpha} \sum_{k=2}^{\infty} (k-1)\, \frac{\alpha^k}{k!} = \alpha - (1 - e^{-\alpha}) = \alpha - p_1(\alpha) = U(\alpha) - 1 \tag{11}$$

$$E[V(\alpha)] = E[U(\alpha)] - 1 = \frac{1}{\alpha}(1 - e^{-\alpha} + \frac{1}{2}\alpha^2) - 1 \tag{12}$$

$$V'(\alpha) = V(\alpha) + \sum_{k=1}^{\infty} \frac{1}{k+1}\, q_k(\alpha) = V(\alpha) + e^{-\alpha} \sum_{k=1}^{\infty} \frac{\alpha^k}{(k+1)!} = \frac{1 - e^{-\alpha}}{\alpha} + \alpha - 1 \tag{13}$$

$$E[V'(\alpha)] = \frac{1}{\alpha} \int_0^{\alpha} V'(a)\, \mathrm{d}a = \frac{\alpha}{4} + \sum_{k=2}^{\infty} \frac{(-\alpha)^k}{(k+1)(k+1)!} = \frac{E_1(\alpha) + \ln(\alpha) + \gamma}{\alpha} + \frac{\alpha}{2} - 1 \tag{14}$$

where

$$E_1(\alpha) = \int_1^{\infty} \frac{e^{-t\alpha}}{t}\, \mathrm{d}t = \int_{\alpha}^{\infty} \frac{e^{-t}}{t}\, \mathrm{d}t \tag{15}$$

is the exponential integral and $\gamma \approx 0.57721\,56649$ is Euler's constant.

$$L(\alpha) = L'(\alpha) = \sum_{k=2}^{\infty} k(k-1)\, q_k(\alpha) = e^{-\alpha} \sum_{k=2}^{\infty} \frac{\alpha^k}{(k-2)!} = \alpha^2 \tag{16}$$

$$E[L(\alpha)] = E[L'(\alpha)] = \frac{1}{\alpha} \int_0^{\alpha} L(a)\, \mathrm{d}a = \frac{1}{3}\alpha^2 \tag{17}$$

Appendix
Equations from [3, p 924]

In [3, p 924], Vitter gives expressions for $U_V(\alpha)$ and $S_V(\alpha)$, which are repeated here as Equations (18) and (23). Vitter does not provide formulae for expected values $E[U_V(\alpha)]$ and $E[S_V(\alpha)]$, which are provided here as Equations (22) and (25).

$$U_V(\alpha) = \begin{cases} U_{V\leq}(\alpha) & \alpha \leq \zeta \\ U_{V\geq}(\alpha) & \alpha \geq \zeta \end{cases} \tag{18}$$

where β is given,

$$\zeta = \lambda\beta = \frac{\lambda}{e^{-\lambda} + \lambda}\,, \tag{19}$$

λ is the unique nonnegative solution of

$$e^{-\lambda} + \lambda = \frac{1}{\beta}\,, \tag{20}$$

and

$$\begin{aligned} U_{V\leq}(\alpha) &= \frac{\alpha}{\beta} + e^{-\alpha/\beta} \text{ and} \\ U_{V\geq}(\alpha) &= \frac{1}{\beta} + \frac{1}{4}(e^{2(\alpha/\beta-\lambda)} - 1)\left(3 - \frac{2}{\beta} + 2\lambda\right) - \frac{1}{2}\left(\frac{\alpha}{\beta} - \lambda\right) . \end{aligned} \tag{21}$$

For purposes of these derivations, let $\mu = \min(\alpha, \zeta)$. Then

$$\begin{aligned} E[U_V(\alpha)] &= \frac{1}{\alpha}\left(\int_0^\mu U_{V\leq}(a)\,\mathrm{d}a + \int_\mu^\alpha U_{V\geq}(a)\,\mathrm{d}a\right) \\ &= \frac{\beta}{\alpha}\left(1 - e^{-\mu/\beta}\right) + \frac{\mu^2}{2\alpha\beta} + \\ &\quad \frac{1}{8\alpha}\left((2\zeta + 3\beta - 2)\left(e^{2\alpha/\beta} - e^{2\mu/\beta}\right)e^{-2\lambda} - \frac{2}{\beta}(\mu + 3\beta - 6 + \alpha)(\alpha - \mu)\right) . \end{aligned} \tag{22}$$

$$S_V(\alpha) = \begin{cases} S_{V\leq}(\alpha) & \alpha \leq \lambda\beta \\ S_{V\geq}(\alpha) & \alpha \geq \lambda\beta \end{cases} \tag{23}$$

where

$$\begin{aligned} S_{V\leq}(\alpha) &= 1 + \frac{1}{2}\frac{\alpha}{\beta} \text{ and} \\ S_{V\geq}(\alpha) &= 1 + \frac{1}{8}\frac{\beta}{\alpha}\left(e^{2(\alpha/\beta-\lambda)} - 1 - 2\left(\frac{\alpha}{\beta} - \lambda\right)\right)\left(3 - \frac{2}{\beta} + 2\lambda\right) + \frac{1}{4}\left(\frac{\alpha}{\beta} + \lambda\right) + \frac{1}{4}\lambda\left(1 - \frac{\lambda\beta}{\alpha}\right) . \end{aligned} \tag{24}$$

$$\begin{aligned} E[S_V(\alpha)] &= \frac{1}{\alpha}\left(\int_0^\mu S_{V\leq}(a)\,\mathrm{d}a + \int_\mu^\alpha S_{V\geq}(a)\,\mathrm{d}a\right) \\ &= \frac{\mu}{\alpha}\left(1 + \frac{\mu}{4\beta}\right) + \\ &\quad \frac{1}{8\alpha}\Bigg(-(2\zeta + 3\beta - 2)\left(E_1\left(-\frac{2\alpha}{\beta}\right) - E_1\left(-\frac{2\mu}{\beta}\right)\right)e^{-2\lambda} \\ &\qquad + (2\zeta\lambda + 4\zeta - 3\beta - 4\lambda + 2)\ln\frac{\alpha}{\mu} + \frac{(\mu + 2\beta + 4 + \alpha)(\alpha - \mu)}{\beta}\Bigg) . \end{aligned} \tag{25}$$

In [3, p 924], Vitter recommends $\beta = 0.86$, from which $\lambda \approx 0.63$ and $\zeta = \lambda\beta \approx 0.542$.