Application of Range Searching to Fast Financial Risk Estimation

Ashok Srinivasan*
UC, Santa Barbara

Viral Shah
PVPCE, Bombay

GVR Phanindra
IIT, Bombay

Ajay Shah
IGIDR, Bombay

Vikram Aggarwal
IIT, Bombay

December 14, 2000

Abstract

Value at Risk (VaR) is a popular measure of the financial risk of a portfolio. Monte Carlo techniques are often used to estimate this quantity. One of the main draw backs of these techniques is the computational cost involved. Under certain conditions, the main bottleneck is a matrix vector multiplication involved in the computation. In the VaR decision problem, we are interested not in the VaR itself, but in whether it exceeds a given threshold. We show in this paper that we can use computational geometry techniques, in particular, range search, to eliminate rows of the matrix to obtain significant increases in speed for a particular class of VaR problems.

*Department of Computer Science, University of California at Santa Barbara, Santa Barbara CA 93106, USA, email: ashok@cs.ucsb.edu

Contents

1 Introduction

The Value at Risk (VaR) is a popular measure of the risk associated with a portfolio. It is the loss expected to be exceeded with a certain probability, usually 1% or sometimes 5%, over a certain time period, say, a day. When Monte Carlo is used to estimate this quantity, the computation involved can be intuitively explained as follows. A "portfolio" consists of different quantities of different "products". It can be represented as a vector w, where each component represents the quantity of the corresponding product in the portfolio. The values of a unit of all the product can similarly be represented by a vector, and the inner product of this vector with w gives the value of the portfolio. If we consider the period of interest to us as a day, then the value of each product will be different at the end of each day. We can model the values of the products at the end of the day as random variables from some probability distribution, and get the joint probability distribution for their values. If we subtract the value of the portfolio at the end of the day from that at the beginning of the day, then we get the loss for that portfolio.

We take random samples from this joint distribution for the products' values, and then determine the loss associated with each sample. If we take N samples, then the $N/100$ th largest loss gives an estimate of the VaR at the 99% (100 - 1%) level of significance. The computation can be abstracted as follows. Determine the VaR as the p th largest component of the vector $v = Lw$, where L is an $N \times M$ matrix when we have N samples and M products (typically, $N >> M$). Here, p is $N/100$ or $N/20$ for 99% and 95% significance, respectively. This is explained precisely in section 2. The corresponding decision problem, which has important practical applications, determines whether or not the VaR exceeds some specified amount, called the "collateral".

Among the different techniques used to estimate VaR, Monte Carlo has the advantage of being flexible and accurate, though the computational effort required for ensuring accuracy can be substantial. In view of the high computational effort required, other approaches involving certain approximations are usually tried. Alternatively, there has been some work [3, 5] done on reducing the computational effort involved in the Monte Carlo technique, which we shall summarize later. We propose a new technique to reduce the computational effort of the VaR decision problem, under certain assumptions, and also suggest an approach to computing the VaR efficiently, by modifying this technique.

The outline of the rest of this paper is as follows. We first define some terms used in this paper, including the precise definition of the *VaR decision* *problem*, in section 2. We then summarize some recent work to reduce the cost of these computations in section 3. We then explain the mathematical basis of our method in section 4, and the computational geometry data structures and algorithms that we use, in section 5. Section 6 describes our tests to evaluate the effectiveness of our technique, and the results of those tests. We conclude in section 7 by summarizing the results and suggesting new directions to improve on the current technique.

2 Notation and definitions

The notation given here is consistent with those in Srinivasan and Shah [5]. Let a portfolio consist of M products, with T different underlyings. That is, a portfolio (or, equivalently, weight vector) is a vector $w \in \mathbb{R}^M$, where a component w_m denotes the quantity of product m in the portfolio, and the value of each product is a function of T (common) variables. Let V_0 be the value of the portfolio at the end of the previous day.

The Monte Carlo estimation is based on sampling random vectors from the joint distribution $f()$ of the underlyings $(r_1, r_2, \ldots, r_T)$. This simulation yields random vectors $r \in \mathbb{R}^T$. This modeling is completely general insofar as models of the returns process are concerned. That is, the r_is can be modeled arbitrarily; the Monte Carlo technique is applicable to almost any type of modeling.

For each product, a valuation function $v(m, r)$ maps r into the value of a product, while $v_o(m)$ is the value of product m the previous day. Again, there is essentially no restriction on how the value of the product is determined.

Suppose a Monte Carlo simulation is based on N draws $r_n \sim f()$. Let "loss matrix" L be an $N \times M$ matrix defined as $L_{nm} = v_o(m) - v(m, r_n)$. The $N \times 1$ vector $b = Lw$ is a set of T draws from the overnight loss on a portfolio w, with a positive component of b indicating a loss, and a negative component a profit. Let V_R be the order statistic estimated from the values of b at a P level of significance: when we desire a VaR estimate with $P = 99\%$ off $N = 15000$ elements in b, we seek the $p = (1 - P)N = 150$ th largest component of b. The desired VaR estimate is V_R. The significance of this quantity is that the loss is expected to exceed the VaR on only 1% of the days.

We focus on one special case: where a large number of VaR problems need to be solved for different portfolios, for a fixed universe of assets with a fixed data generating process (implying a fixed L matrix). This is an *important special case in the real–world. See reference [5] for details, such as applications of this problem. Often, one does not deal with such a situation; we may just need to evaluate a single portfolio with a single L matrix. Our technique is not helpful in this case.*

The VaR Decision Problem. There are situations where knowing VaR is in itself important. However, in other situations such as the risk containment of a futures clearing corporation, it is more important to *check whether the magnitude of VaR exceeds "collateral"*. We call this the "VaR decision problem":

> Given the loss matrix L, the collateral value V_c, and the weight (portfolio) vector w, is $V_R > V_c$, where V_R is the $p = (1 - P)N$ th largest component of Lw?

3 Current computational techniques

Conventional technique. The conventional method for solving this is to actually evaluate VaR(w) and then check if it exceeds V_c as shown in Fig 3.1. In this, the major cost of computation is in the evaluation of the matrix–vector product Lw. This takes $O(NM)$ time, where typically $N >> M$. The next step is to find the p th largest component of b. This can be found in $O(N)$ time using a selection algorithm. We thus get a total complexity of $O(NM)$, with the matrix-vector multiplication being the dominant factor.

Algorithm 3.1 Conventional technique for computing VaR

Input: $L \in \mathbb{R}^{N \times M}$, $w \in \mathbb{R}^M$ and $V_c > 0$
Output: Answer to the decision problem: YES or NO

```
1: Compute b = Lw in O(NM) time.
2: Determine V_R, the p th largest component of b in O(N) time using a
   linear time selection algorithm.
3: if V_R > V_c then
4:    Output YES
5: else
6:    Output NO
7: end if
```

Different techniques can be applied to reduce the time required for this computation. For example, we can use a low discrepancy sequence to generate (conceptually) the loss matrix [3], leading to an L matrix with fewer rows than if a pseudo-random sequence were used, to get the same accuracy.

Srinivasan and Shah [5] take a different approach. Given the loss matrix L, whether generated using low-discrepancy sequences or pseudorandom sequences, it is preprocessed so that an upper bound on the VaR can be computed fast, without computing the actual VaR. If the bound does not exceed the collateral, then the decision problem can be answered as *NO*. If the bound exceeds collateral, then certain heuristics are used to determine the exact answer.

4 Mathematical basis of the range search technique

We next describe the mathematical basis for our technique. As in [5] our technique is based on the following observation:

- The matrix L is constant throughout the day. We can therefore afford to spend a large amount of computational effort in analyzing it initially, if this will result in answering the decision problem quickly when presented with a weight vector.

In the conventional technique of Fig 3.1, an inner product of w is computed with each row of L to produce the resulting vector b, and then the p th largest component is obtained. Our approach is based on eliminating those rows of L that we know will result in an inner product smaller than V_c, so that we need to determine inner products for fewer rows, as explained below.

Srinivasan and Shah [5] suggest the following technique: We wish to check if $V_R > V_c$. Equivalently, we wish to check if there are p components of Lw with magnitude greater than $|V_c|$ and a positive sign. We, therefore, do not need to compute those components that we know will have a magnitude smaller than V_c. Using the Cauchy-Schwartz inequality [1, 2], we can ignore rows of L with L_2 norm less than $|V_c|/\|w\|_2$. We compute the inner product with w for only those rows of L having a greater magnitude, and then count the number of components greater than V_c. If that number is at least p, then the answer to the decision problem is *YES*. Otherwise, it is *NO*.

The is illustrated in Algorithm 4.1. This technique, as implemented in [5], actually resulted in an increase in the time taken! The reason for this was conjectured to be the creation of temporary arrays in the Matlab implementation. In the current work, we implemented this technique in C and observed a significant speed-up compared with the conventional technique. However, we went one step further, as explained below.

If $L = UDV$ is the singular value decomposition of L, then the algorithm 4.1 can be applied to $U, (DVw)$, instead of to L, w. Note that DV can be precomputed, and only the M non-zero rows retained, to get an $M \times M$ matrix. Consequently, we need to compute and store only the corresponding M columns of U. In the rest of this paper, we refer to only such a "reduced" singular value decomposition of L. That is D refers to the $M \times M$ submatrix with only non-zero rows (if the rank of L is M), and U refers to the the corresponding $N \times M$ submatrix.

In the approach we propose here, we combine the information from both the alternatives: that is, we need to consider only those rows of L that are accepted using Algorithm 4.1 applied using both L, w *and* $U, (DVw)$. We use range–searching to determine this set of rows efficiently.[1] This is given in Algorithm 4.2.

5 Data structures and algorithms

Let P_t be the set of N points with the L_2 norms of rows of L being the x coordinates, and the L_2 norms of rows of U being the y coordinates. We use the range tree data structure to retrieve only those rows corresponding to points $p = (x, y)$ with $x > V_c/\|w\|_2$ *and* $y > V_c/\|\hat{w}\|_2$, where $\hat{w} = DVw$. The details of this data structure are given in Preparata and Shamos [4]. We summarize it below and then mention specific features of our implementation.

The primary structure of a range tree is a segment tree S_p, which is a balanced binary search tree, with each node representing an interval on the x-axis. The interval represented by a parent node is the disjoint union of the intervals represented by the children, and the root represents the interval between the extremes of the x coordinates of the points in P_t. The segment tree is used here to search for the x-coordinates of points. Any interval along the x-axis can be represented as the disjoint union of $O(\log N)$ nodes of S_p. Let $P_t(\nu) \in P_t$ contain those points in P_t lying in the interval represented by a node ν of the segment tree. Each node ν points to a secondary data structure $S_s(\nu)$ which stores $P_t(\nu)$. In typical implementations,

[1] We can, alternatively, use a brute-force approach that checks each row under both these criteria. Though this will result in an improvement to speed compared with using just one of the two pieces of information, it is not as efficient as normal range search, in a general situation.

Algorithm 4.1 Eliminating rows with "small".

Input: $L \in \mathbb{R}^{N\times M}$, $w \in \mathbb{R}^{M}$, and collateral V_c.
Output: An answer to the decision problem: YES or NO

- Initialization:

  Compute the L_2 norm of each row of L.
  Sort the rows of L in descending order of their L_2 norms and store them in a matrix $\tilde{L}$. If $\tilde{L}_i$ denotes the i th row of $\tilde{L}$, then $i < j$ implies $\|\tilde{L}_i\|_2 \geq \|\tilde{L}_j\|_2$ since the rows are sorted.
  Store the corresponding norms of the rows in a sorted array $\tilde{l}$.

- Computation:

  Compute $\|w\|_2$ in $O(M)$ time.
  Determine the largest index s such that $\tilde{l}_s > |V_c|/\|w\|_2$.
  if there is no such component **then**
    set $s = 0$.
  end if This takes $O(\log N)$ time using binary search.
  if $s < p$ **then**
    Output NO
  else
    Compute inner product $\tilde{L}_i w$ for each $i \leq s$ and call the set of inner products computed $\hat{b}$. This step takes $O(sM)$ time.
  end if
  if the number of elements in $\hat{b}$ greater than V_c is fewer than p **then**
    Output NO.
  else
    Output YES
  end if This can be done in $O(s)$ time.

Therefore the total time taken is $O(sM + \log N)$. In the worst case, the time taken can be $O(NM)$. However, if s is small, then there is a potential for a significant reduction in time.

Algorithm 4.2 Eliminating rows with "small" norms norms using two types of information.

Input: $L \in \mathbb{R}^{N \times M}$, $w \in \mathbb{R}^M$, and collateral V_c.
Output: An answer to the decision problem: YES or NO

- Initialization:

  Compute the L_2 norm of each row of L.
  Compute the "reduced" singular value decomposition UDV of L.
  Compute $D_V = D * V$.
  Compute the L_2 norm of each row of U.
  Define P_t as the set of N points with the L_2 norms of rows of L being the x coordinates, and the L_2 norms of rows of U being the y coordinates.
  Construct a range-tree R from P_t. The range–tree data structure is summarized in section 5.

- Computation:

  Compute $\|w\|_2$ in $O(M)$ time.
  Compute $\hat{w} = D_V w$ in $O(M^2)$ time.
  Determine all the rows with norm in the L matrix greater than $V_c/\|w\|_2$ *and* norm in the U matrix greater than $V_c/\|\hat{w}\|_2$, using the range tree R, in $O(\log^2 N + s)$ time, where the number of rows retrieved is s. This algorithm is outlined in section 5.
  if $s < p$ **then**
    Output NO
  else
    Compute inner product $L_i w$ for each row retrieved in the search, and call the set of inner products computed $\hat{b}$. {This takes $O(sM)$ time.}
  end if
  if The number of elements in $\hat{b}$ greater than V_c is fewer than p **then**
    Output NO
  else
    Output YES.
  end if This can be done in $O(s)$ time.

The total time taken is $O(M^2 + sM + \log^2 N)$. In the worst case, the total time taken can be $O(NM)$. However, if s is small, then there is a potential for a significant reduction in time.

$S_s(\nu)$ is a balanced binary search tree on the y-coordinates of $P_t(\nu)$. However, we use an array sorted on the y coordinates instead. While the time complexities for searches using either secondary data structure are identical, actual implementations can be expected to perform better with an array; hence our choice. The algorithm for constructing the range tree is outlined in Algorithm 5.1. Details can be found in Preparata and Shamos [4]. The time complexity for creating the range tree is $O(N \log N)$, and the space complexity too is $O(N \log N)$.

Our implementation of the range tree deviates from the standard one in the following ways:

- We use a sorted array as the secondary structure, instead of a balanced binary search tree.
- We need to consider details, such as points not being in general positions, in particular, with the possibility of points being duplicated.
- Our search is bounded on only the lower sides in both the coordinates, and so we adapt the conventional algorithm for this case.

Algorithm 5.1 Creating a Segment tree.

Input: A set P_t of N points in the 2-D plane.
Output: A range tree R representing N points.

- Initialization:
  1: Create the segment tree S_p tree with N leaves, on the x-coordinates of the points in P_t. Each child node is formed by partitioning the interval represented by the parent node, at the median of the x-coordinates of the subset of P_t contained in the parent's interval. The root represents the entire interval between the extreme x-coordinates of the points in P_t. At every node, allocate memory for the array used to store the points in that interval.
  2: **for** each point $p_i = (x_i, y_i) \in P_t$ **do**
  3: Insert p_i recursively at every level of the tree, in the sub-tree corresponding to its interval. The point is placed in the appropriate position in the secondary structure, the array sorted on y_i.
  4: **end for**
  5: The resulting structure is the range tree R.

The procedure for searching the segment tree is outlined in Algorithm 5. We find $O(\log N)$ nodes such that the disjoint union of the intervals represented by those nodes gives the x interval of the range-query. This takes $O(\log N)$ time. We perform a binary search on the secondary structure of each of these nodes to locate the points in the query range. This takes $O(\log N)$ time on each node. Retrieving s points then takes $O(s)$ time. Hence, the time complexity of the search is $O(\log^2 N + s)$.

Algorithm 5.2 Range Search on the Segment tree to find favorable rows.

Input: A point $p = (x, y)$.
Output: All points p_i with $x_i > x$ and $y_i > y$.

- Computation
  1. Recursively find the $O(\log N)$ nodes in S_p representing the interval (x, ∞).
  2. In each such node ν, use binary search on $S_s(\nu)$ to locate the smallest $\hat{y} > y$. Return all points $p_i = (x_i, y_i)$ from $S_s(\nu)$ with $y_i \geq \hat{y}$.

Another possible data structure that could be used for range-searching is a two-dimensional k-d tree. However, its time complexity of $O(\sqrt{N} + s)$ is asymptotically higher. A priority tree could also be used for range-searching. It will be more efficient than a range tree, as its time complexity is $O(\log N + s)$ and space complexity is $O(N)$.

6 Empirical studies

We have mentioned three algorithms for answering the VaR decision problem above. Next we need to compare their relative effectiveness in realistic situations. We first describe our testing procedure and then we present the results on the effectiveness of the different procedures.

The tests are performed on eleven sample L matrices and 1000 random portfolios each. Appendix C of Shah and Srinivasan [5] documents how the eleven L matrices were created. All the code is in C, compiled using *gcc* specifically for the pentiumpro architecture with optimization flags *O3, funroll–all–loops* , and run on an Intel Celeron 566MHz processor with 128MB RAM. The tests were timed with the Standard C library function *clock()* and run multiple times to get enough resolution on the timings.The collateral value was selected by using the heuristic that V_c was 10% of the absolute value of the portfolio at the beginning of the day.

Algorithms Tested: The different algorithms that we bench-marked are as follows:

1. The conventional technique, Algorithm 3.1.
2. Row reduction using only 1 dimension, Algorithm 4.1
3. Row reduction using 2 dimensions, that is, range searching as in Algorithm 4.2

In order to also verify correctness of our implementation, we first compute the VaR estimate with a crude technique, where we replace step 2 of Algorithm 3.1 with first sorting Lw and then selecting the p^{th} largest component. The results of the decision problem solved using this crude technique is then compared with the results obtained from each algorithm, to verify correctness of the implementation.

The test results are presented in Fig 1. In each test, we do not measure the initialization time, as initialization is performed only once. The computation is performed on 1000 portfolios for each of the 11 L matrices.

The timing for the conventional algorithm is shown in the lightest shade of gray[2], and those for one and two dimensional range searching are shown in increasingly darker shades respectively. We can observe that in all the cases, the two-dimensional range search is significantly faster than the other two algorithms.

We also noticed that in both the schemes using row elimination, the dominant factor affecting the speed was the computation of the inner products, and not the search involved in the row elimination process; the time taken for the search is negligible.

7 Conclusions

We have suggested an algorithm to solve the VaR decision problem efficiently using range searching in two dimensions, when the matrix L is held fixed. We have compared it with the conventional technique of performing an entire matrix–vector multiplication, and also with the using a one dimensional

[2] The time taken for the conventional technique is denoted by the upper end of this region, rather than by the length of the region

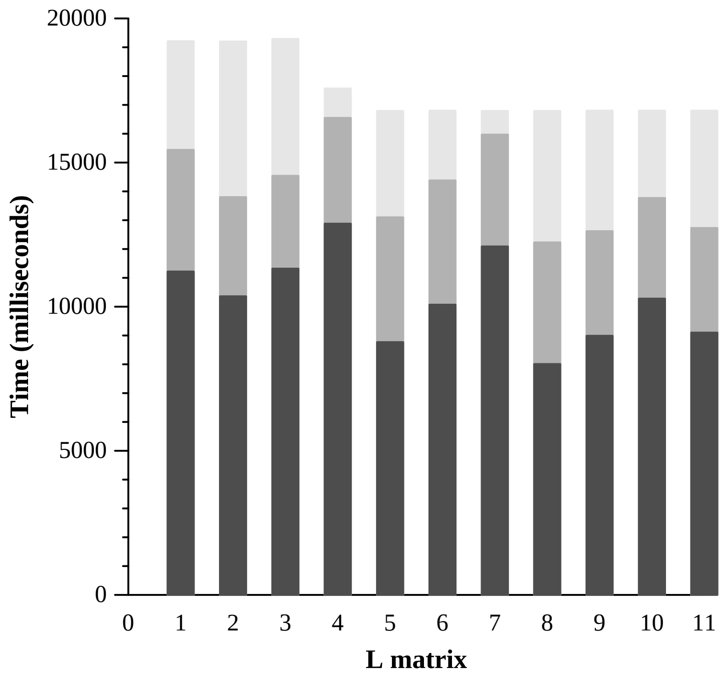

Figure 1: Average run time of each algorithm on 1000 portfolios, for each of the 11 L matrices. The timing for the conventional algorithm is marked by the upper end of the region in the lightest shade of gray, and those for one and two dimensional range searching are shown in increasingly darker shades.

range search, using only norms of the rows of the matrix L. (Using the norms of the rows of U alone will result in a similar computational effort) We have demonstrated that this technique reduced the time taken by a factor of around two, compared with using the conventional technique, and by around 35% compared with a one–dimensional search.

There are further improvements that can be made to increase the speed. Instead of using the Cauchy–Schwartz inequality, we can, in general, use Holder's inequality and choose possibly different norms for the rows and for w to get better results. We observed empirically that the relative magnitudes of the norms of the rows of L and U were somewhat correlated. This implies that U does not give as much additional information as we desire. We can choose other transformations so that the information obtained is substantially different from that of L. We can also incorporate additional information and perform a three dimensional range search. However, there is a limit to the number of "information-dimensions" we can use with range searching, since its performance degrades in high dimensions. An alternative is to use range-search for important dimensions, and then the brute force checking for the other dimensions.

Another line of study will be to solve the non-decision version of this problem, when the actual VaR needs to be estimated. Srinivasan and Shah [5] give expressions for upper bounds on the VaR. If lower bounds are used instead, then the techniques of this paper can be used with lower bounds replacing V_c. If approximate solutions are needed for this problem, then we can consider only those columns of U that are associated with large singular values. That is, we equate the components of D with very small magnitudes to zero. Yet another possibility is to use Monte Carlo techniques yet again in the matrix-vector multiplication.

We wish to note that the techniques proposed above are applicable only when the L matrix is held fixed. If this is not the case, then we need to consider alternatives, such as the use of Quasi-Monte Carlo to reduce the size of the L matrix, thereby reducing the computational complexity.

Acknowledgment: The authors wish thank Shiv Chandrasekaran for fruitful discussions with him, and for his suggestions.

A Notation

The important notations used in this paper are listed in Table 1.

b	Lw.
D	The singular value decomposition of L is $L = UDV$.
f	Joint distribution of the underlyings.
l	A sorted array of the L_2 norms of the rows of L.
L	Loss matrix, $\in \mathbb{R}^{N \times M}$.
L_i	Row i of L.
M	Number of products.
N	Number of draws in the Monte Carlo simulation.
ν	A node of the segment tree S_p.
P	Level of significance for VaR.
p	$(1 - P) \times N$; additionally used to denote a point.
P_t	A set of points.
$P_t(\nu)$	A subset of P_t. Points in $P_t(\nu)$ are in the interval represented by node ν.
r	Vector of values of the underlying.
R	A range tree.
S_p	A segment tree, the primary data structure for the range-tree.
S_s	Secondary data structure for the range tree.
T	Number of underlyings.
U	The singular value decomposition of L is $L = UDV$.
V	The singular value decomposition of L is $L = UDV$.
v	Valuation function for products as a function of the underlyings.
v_o	Value of a product at the beginning of the day.
V_0	Value of a portfolio based on the prices at the beginning of the day.
V_c	Value of the collateral.
V_R	VaR estimate.
w	A portfolio, a vector in $\mathbb{R}^N$.

Table 1: Important notations used in this paper.

References

[1] Friedberg, S. H., Insel, A. J., and Spence, L. E. *Linear Algebra*, second ed. Prentice Hall, New Jersey, 1989.

[2] Heath, M. T. *Scientific Computing: An introductory survey.* McGraw–Hill, New York, 1997.

[3] Papageorgiou, A., and Paskov, S. Deterministic simulation for risk management. *Journal of Portfolio Management* (April 1999).

[4] Preparata, F., and Shamos, M. I. *Computational Geometry: An Introduction.* Prentice Hall, New York, 1985.

[5] Srinivasan, A., and Shah, A. Improved techniques for using Monte Carlo in VaR estimation. *Submitted to the National Stock Exchange Research Initiative* (2000).