Speeding Up Two-Level Simulation for Tail Conditional Expectations by Means of Prefix Sum Based Algorithms

Yi-Cheng Tsai, Hsin-Tsung Peng, Jan-Ming Ho

Bryant Chen, and Ming-Yang Kao

***Abstract*—In this paper, we study the problem of computing tail conditional expectations (*TCE*) of portfolio gains at specific times (*T*) in the future. We present efficient algorithms to handle the following two cases: (1) we have only one option (call or put) in our portfolio, denoted as a single-stock-single-option portfolio (SSSO); (2) we have a stock and some of its options in our portfolio, denoted as a single-stock-multiple-options portfolio (SSMO). Compared with previous simulation algorithms, our algorithms compute *TCE* of a given portfolio more efficiently and still maintain the same degree of accuracy. In the SSSO case, we reduce the computational time complexity from the SSSO-Naive Algorithm's $O(m*n)$ to SSSO algorithm's $O(m+n)$, where m is the number of possible price outcomes for an underlying stock at time T and n is the number of possible price outcomes for an underlying stock at time U (maturity) in respect to each possible price outcome for an underlying stock at time T. In the SSMO case, we provide two algorithms to compute the *TCE*. The computational time complexity of the SSMO-Naive Algorithm is $O(q*m*n+m*\log(m))$, where q is the number of options. The computational time complexities of our two algorithms are $O(q*(m+n)+m*\log(m))$, and $O(q*\log(q)+n+m*q+m*q*\log(q)+m*\log(m))$ or $O(q*\log(q)+n+m*q+m*q*\log(m)+m*\log(m))$. In both cases, our experiments show that when m and n are greater than five thousand, our algorithms run thousands of times faster than the naive algorithms.**

***Index Terms*—Portfolio, Tail Conditional Expectations, Risk Management, Risk Measurement, and Option**

I. INTRODUCTION

The current U.S. subprime mortgage crisis has highlighted the importance of risk management in finance. In this global financial crisis, many banks and other financial institutions around the world have suffered huge losses, many of which might have been avoided by better risk management.

Considerable research has been done on developing methods for risk management. Evaluating investment risk accurately specifically is a core issue within this research. The value-at-risk (*VaR*) [1] is a well-known risk measure used for evaluating investment risk. In recent years, coherent risk measures have received attentions increasingly. According to [2, 3], coherent risk measures must satisfy four properties, namely, translation invariance, subadditivity, positive homogeneity, and monotonicity. The tail conditional expectation (*TCE*) proposed in [3] is one such coherent risk measure and is closely related to conditional value-at-risk [4].

Recently, there has been a growing interest in the use of *TCE* in measuring risk. Landsman et al. [5, 6] derived explicit formulas for computing *TCE* for the elliptical distribution and derived the *TCE* for loss random variables in the exponential dispersion models. Furman et al. [7] calculated the TCE for a multivariate gamma portfolio of risks based on its non-negative risks allocation. Brazauskas et al. [8] presented an estimator for estimating the *TCE* and constructed its confidence intervals and bands. Cai et al. [9] derived various *TCEs* for multivariate phase-type distributions by using the Markovian method and focused on the risk of an entire portfolio. These researches focus on the *TCE* estimation of some specific distributions of portfolios. However, there are still some portfolios (such as the portfolios of options) require using two-level simulation to obtain the portfolios gains. In our paper, we will study the computation of the *TCE* of portfolios containing options.

Let V be a portfolio's possible gain at a specified time T in the future. The *VaR* will then be defined as the negative of the p-quantile of the cumulative distribution function of V. The TCE_p is defined as:

$$TCE_p := E\left[-V \mid V \le -VaR_p\right] \quad (1)$$

Since risk measures typically have to deal with thousands of probable outcomes of multiple assets in a portfolio, computing them in naive manner often takes a prohibitive amount of computational time. Moreover, the portfolios calculated by two-level simulation for their gains make the computation of their *TCE* more complex and time-consuming. So, some works are done to investigate how to speed up the computation of risk measures. Lan et al. [10] proposed an efficient method to construct a confidence interval for TCE_p to speed up the computation of the *TCE* estimation of one option.

In this paper, we will provide efficient algorithms to compute the *TCE* value itself of portfolios containing options in two cases. In the first case, we have only one option (call or put) in our portfolio, called a single-stock-single-option

Yi-Cheng Tsai and Hsin-Tsung Peng are research assistants, and Jan-Ming Ho is Research Fellow in the Institute of Information Science, Academia Sinica, Taipei, Taiwan. Their e-mails are {yicheng, m9115013, hoho}@iis.sinica.edu.tw.

Bryant Chen is a research assistant in the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His e-mail is bryantrchen@gmail.com.

Ming-Yang Kao is Professor of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL 60201, USA. His e-mail is kao@northwestern.edu.

portfolio (SSSO); in the second case, we have a stock and some of its options in our portfolio, called a single-stock-multiple-options portfolio (SSMO). Two key ideas used in our algorithms are (1) an algorithmic technique called prefix sum computation and (2) the standard model assumption that the distribution of the stock price S_U at a time U relative to the stock price S_T at a time $T < U$ is identical for all S_T. Our algorithms avoid a significant amount of redundant computation by applying the prefix sum technique to take advantage of the identical distribution assumption. We can adapt our algorithms for a relaxation of the identical distribution assumption at the cost of an increased running time.

The remainder of the paper is organized as follows: In Section II, we describe the background of our research. In Section III, we provide a formal definition of the problem in the paper. In Sections IV and V, we present our algorithms for SSSO and SSMO, respectively. In Section VI, we describe our experiments. Lastly, in Section VII, we summarize our findings and suggest future research directions.

II. Background

To calculate the *TCE*, we need a method for pricing options at time 0 and time T. There are many widely used pricing models for various derivatives. In this paper, we use the Black-Scholes model [11]. We review this model for a put option as follows:

$$P_0 = Ke^{-rU}\phi(-d_2) - S_0\phi(-d_1), \tag{2}$$

$$P_T = Ke^{-r(U-T)}\phi(-d'_2) - S_T\phi(-d'_1), \tag{3}$$

where

$$d_1 = \frac{\ln(S_0 / K) + (r + \sigma^2 / 2)U}{\sigma\sqrt{U}}, \tag{4}$$

$$d_2 = d_1 - \sigma\sqrt{U}, \tag{5}$$

$$d'_1 = \frac{\ln(S_T / K) + (r + \sigma^2 / 2)(U - T)}{\sigma\sqrt{U - T}}, \tag{6}$$

$$d'_2 = d'_1 - \sigma\sqrt{U - T}. \tag{7}$$

In these equations, φ is the cumulative normal distribution, r is the non-risk interest rate, K is the strike price of the option, S_0 is the initial price of the underlying stock, and σ is the volatility of the underlying stock. We can also obtain the value of the stock price at time T (*i.e.*, S_T) and time U (*i.e.*, S_U) through the geometric Brownian motion.

$$S_T = S_0 \exp((\mu - \frac{\sigma^2}{2})T + \sigma\sqrt{T}Z), \tag{8}$$

$$S_U = S_T \exp((\mu - \frac{\sigma^2}{2})(U - T) + \sigma\sqrt{U - T}Z'). \tag{9}$$

Z and Z' above are the standard normal random variables at time T and U respectively. We can then compute TCE_p by numerical integration using the equations described above. The result obtained serves as the benchmark for our experiment in the SSSO case.

Since it is time-consuming to use the Black-Scholes formula to calculate the *TCE*, we thus use a discrete model to approximate the *TCE*. For our experiments, we chose to use the binomial model developed by Cox, Ross, and Rubinstein [12]. The binomial model assumes that at each time interval, the underlying stock price can only go up or down. Let u be the ratio of up movement for a stock after each interval and let d be the ratio of down movement after each interval. Then the values of u and d are:

$$u = e^{\sigma\sqrt{t}}, \tag{10}$$

$$d = e^{-\sigma\sqrt{t}}. \tag{11}$$

The parameter t here is the ratio of the time period of one interval to the time period for which σ is calculated. The probability of the stock price moving higher is:

$$p = \frac{e^{r(t)} - d}{u - d}. \tag{12}$$

Now we can obtain the value of each possible stock price at all time points in the binomial tree.

The price of a put option at time T can be computed by the equation below:

$$P_T = e^{-r(U-T)} \sum_{i=0}^{n} \left(\frac{n!}{i!(n-i)!} \right) p^i (1-p)^{n-i} (K - S_T u^i d^{n-i})^+ \tag{13}$$

In Equation (13), n is the number of steps in the binomial tree. After we obtain the option prices of each node at time T and U, we compute the *TCE*. This algorithm is referred to as SSSO-Naive algorithm. Note that computational time complexity of SSSO-Naive algorithm is $O(m*n)$.

III. Problem Definition

First, we assume that we can obtain the distribution $F_S(t)$ of the future price of a stock at time t, $0 \leq t \leq U$, where $F_S(t)$ can be either empirically or theoretically computed Note that U denotes the striking time of the option under consideration. We are interested in computing *TCE* at time T, where $0 < T < U$. Let $F_S = F_S(T)$ be the distribution of stock price S at time T and F_R be the distribution of stock price ratio $R = F_S(U)/F_S(T)$ at time U with respect to time T.

Assume that at time T, there are h assets in our portfolio and each asset has m possible price outcomes for the underlying stock F_S and that for each of the m values of F_S at time T, there are n possible price outcomes at time U. Under these assumptions, a naive algorithm, which takes every possible outcome into account, would require a computational time complexity computation on the order of $O(m*n*h)$. It would take as long as a half of a day to compute the *TCE* value of just one option, the smallest possible h, if m and n were greater than a hundred thousand. Our algorithms achieve the same accuracy as the naive algorithms in only a few seconds.

In order to reduce computation time, we design algorithms that greatly avoid a considerable amount of redundant computation without sacrificing accuracy. The details of each case will be described as follows.

IV. Single-Stock-Single-Option Portfolio (SSSO)

In SSSO, we have only one option (call or put) in our portfolio. We chose "sell put" for this experiment case. The gain of a put option is determined by the strike price minus the price of the underlying stock at maturity. If the price of the underlying stock at maturity is greater than the strike price, the value of the option is zero. In this section, we first

introduce how we used the naive algorithm to compute TCE_p. We then present our algorithm for computing TCE_p in SSSO, which we will call "SSSO Algorithm".

A. SSSO-Naive Algorithm

We are interested in the computation of the TCE_p at time T, where S is the stock price, P_0 is the initial option price, U is the maturity, K is the strike price, and r is the interest rate. Let the number of nodes at time T and time U be m and n respectively.

The stock price at time T is S_i, $i = 1, 2,\ldots, m$, and the stock price ratio is R_{ij}, $j= 1, 2,\ldots, n$. The period from each possible price outcomes at time T to U is the same, so the stock price ratio in respect to each possible price outcomes is the same. Therefore we can use R_j to express the stock price ratio in respect to each possible price outcomes. Because the possible prices of underlying stock on binomial tree have been sorted by their values, the possible gains of the portfolio in this case are sorted too. Take selling a put option for example, if the possible underlying stock prices strict increase, the possible gains of selling a put option will strict increase. Then, we can find out the position of p-quantile among the nodes at time T before calculating the portfolio gain. After determining the position of p-quantile, we can screen out the points that are greater than the p-quantile in the tree at time T. If $K \geqq S_i*R_j$, portfolio gain (v) equals

$$v = e^{-r(U-T)}(P_0 e^{rU} - (K - S_i * R_j)) \quad (14)$$

If $K < S_i*R_j$, portfolio gain (v) equals

$$v = e^{-r(U-T)} P_0 e^{rU} \quad (15)$$

The portfolio gain at time T can be computed by

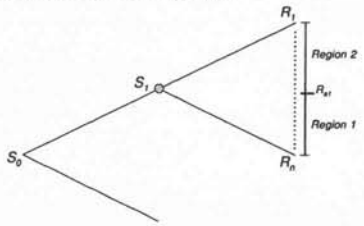
$$V_i = E[v \mid S_i] \quad (16)$$

Then we can get TCE_p by Equation (1). The computational time complexity of the SSSO-Naive Algorithm is $O(m*n)$.

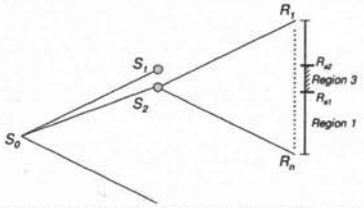
B. SSSO Algorithm

We can reduce our computation time if we can find the value of K/S_i in F_R rapidly. K/S_i, for the purposes of this paper, will be called the strike price ratio.

We can derive these two inequalities from the binomial model: $S_1 \geq S_2 \geq \ldots \geq S_m$ and $R_1 \geq R_2 \geq \ldots \geq R_n$. From the first inequality, we obtain that the strike price ratio K/S_i is a monotonic series, $K/S_m \geq K/S_{m-1} \geq \ldots \geq K/S_1$. Now, we want to find the point that is closest and smaller than K/S_i in F_R, and we will use R_{ai} to represent those points, where i=1,2,…,m. The strike price ratio is a monotonic series, so is the series R_{ai}. First, we use linear comparison to find out the closest point of K/S_1 in F_R showing as the point R_{a1} in Fig. 1(a). Because monotonicity of R_{ai}, we do not need to compare K/S_2 and R_j from the beginning. However, it is necessary to compare the points that are equal or greater than R_{a1}. We will save the results of R_{ak} for the searching of $R_{a(k+1)}$. Take the computation of V_1 and V_2 for example, we show that SSSO Algorithm uses prefix sum method to reduce the computation. After determining the position of the R_{a1}, we can divide F_R into two regions: The first region is the points equal or lesser than R_{a1} in F_R. The second region is the points greater than R_{a1} in F_R., as shown in Fig. 1. In Fig. 1(a), we can use Equation (14) to compute the possible gains in region 1and Equation (15) to compute the possible gains in region 2, then use equation (16) to compute V_1. When we want to compute V_2 in Fig. 1(b), we do not need to compute the points in region 1 again. We merely need the sum of the points in region 3 and the results from region 1to obtain V_2. Using this prefix sum computation, we only need to go through F_R once, and we can obtain the values from V_1 to V_m. In the same fashion as the naive algorithm, we can screen out the points that are greater than the p-quantile in the tree at time T. After calculating all the values of the portfolio gain in the p-quantile region, we can get TCE_p by Equation (1). The computational time complexity of the SSSO Algorithm is $O(m+n)$.



(a) In S_1, strike price ratio (K/S_1) divides F_R into two regions.



(b) In S_2, we only need to compute the region 3 when the region 1 was calculated in S_1.

Fig. 1: Prefix Sum Computation.

V. Single-Stock-Multiple-Option Portfolio (SSMO)

In SSMO, there is one stock and multiple options (buy call, sell call, buy put or sell put four kinds of trading) of that stock in our portfolio. We assume that there are q options and that the weight of each option is equal. We will first discuss the naive algorithm in SSMO and present two algorithms to handle this case, comparing the time complexity with SSMO-Naive Algorithm.

A. SSMO-Naive Algorithm

Because there are q options in our portfolio, we need to repeat the SSSO-Naive Algorithm q times to get probable gains for every option at time T. We name the probable gains for every option V_{ki} where $k = 1, 2,\ldots,q$ and $i = 1, 2,\ldots,m$. Then, we can compute the probable gain of our portfolio at time T by summing up the gain provided by each option, which is $V_i = \sum_{k=1}^{q} V_{ki}$. After obtaining the portfolio gain V_i, we sort V_i and find the p-quantile and compute the TCE_p as described in SSSO. The computational time complexity of the naive algorithm is $O(q*m*n+m*\log(m))$.

B. Two Algorithms for SSMO

We present two algorithms to handle SSMO. The first will be called the q-times-SSSO Algorithm, and the second will be called the SSMO Algorithm.

1) q-times-SSSO Algorithm

We use SSSO algorithm to compute V_{ki}. Then, we calculate the TCE_p value by the process described in naive algorithm of SSMO. The computational time complexity of q-times-SSSO Algorithm is $O(q*(m+n)+m*\log(m))$.

2) SSMO Algorithm

In SSMO, there is only one underlying stock of the options in our portfolio and its initial price is S_0. Let the strike price of kth option be K_k, $k = 1, 2,\ldots, q$. We sort K first, so $K_1 \geqq K_2 \geqq \ldots \geqq K_q$; let $A = \{a_{ik}\}$, $a_{ik} = K_k/S_i$, $i = 1, 2,\ldots,m$; $S_1 \geqq S_2 \geqq \ldots \geqq S_m$. We can then see that every row and every column in the matrix A is sorted, so we can use a merge sort to sort every element in the matrix A to form a non-decreasing vector B. The probability distribution of price change ratio at time U R_j, $j = 1, 2,\ldots,n$, is also a sorted vector. A merge sort can then be done again to combine vector B and vector R and form a non-decreasing vector C. As described in Section IV SSSO Algorithm, when we have the position of the strike price ratio in the distribution F_R, we can use Equation (14) and (15) to compute V_i. Therefore, we only need to go through vector C once to obtain the portfolio gain $V_1, V_2,\ldots, V_m$.

Similar to SSMO-Naive Algorithm and q-times-SSSO Algorithm in this section, we just need to follow the same process to obtain the TCE_p value. The computational time complexity of the SSMO Algorithm is $O(q*\log(q)+n+m*q+m*q*\log(q)+m*\log(m))$ or $O(q*\log(q)+n+m*q+m*q*\log(m)+m*\log(m))$.

The difference of these two computational complexities comes from which direction (row or column) you choose for the merge sort. We will compare the value of q to the value of m. If m is bigger than q, we will choose the algorithm of $O(q*\log(q)+n+m*q+m*q*\log(q)+m*\log(m))$ computational time complexity. If q is bigger than m, we will choose the other one.

VI. Experiments

In order to analyze the property of our algorithms in both cases, we conducted a few experiments where we calculated the accuracy of our algorithms and compared the computation time to the naive algorithms of both cases. Our facility is equipped with an IBM XSERIES_3455 loaded with 10 GB of RAM. The programs are written by using Perl language.

A. Experiment Setting

In SSSO, we sell a put option in our portfolio. The initial price is $S_0 = 100$; the strike price of the put option is $K = 110$; the maturity is $U = 1$ year; and the volatility of the underlying stock is $\sigma = 15\%$; the interest rate is $r = 6\%$.

In SSMO, the parameters of the underlying stock are the same as those in SSSO. The strike prices of the options in SSMO are random numbers and the value of the strike prices is lesser than $S_0*(1+20\%)$ and greater than $S_0*(1-20\%)$. The maturity of all options is U. In both cases, we are interested in calculating the TCE_p, where $p = 1\%$ at time $T = 1/52$ year.

In our experiment, we use two metrics to evaluate the performances of our algorithms as follows.

$$error\ rate = \left| \frac{TCE_{p\_SSSO\ Algorithm} - TCE_{p\_benchmark}}{TCE_{p\_benchmark}} \right| \times 100\%. \quad (17)$$

$$accuracy = 1 - error\ rate \quad (18)$$

where $TCE_{p\_SSSO\ Algorithm}$ is the TCE_p value calculated by SSSO algorithm, and $TCE_{p\_benchmark}$ is the TCE_p value calculated by Black-Scholes formula.

B. Experiment Results

1) SSSO

For this case, we used Black-Scholes formula to calculate $TCE_{0.01}$, and the result was 3.39148. We then used this value as $TCE_{p\_benchmark}$ in Equation (17) to analyze the accuracy of our algorithm. We repeated every experiment five times and took the average for the result. In Fig. 2, we fixed the value of n to 10000. We observed that the error rate of $TCE_{0.01}$ decreases as m increases from 100 to 10000. When m is greater than 5000, the accuracy of SSSO Algorithm rises above 99.5%.

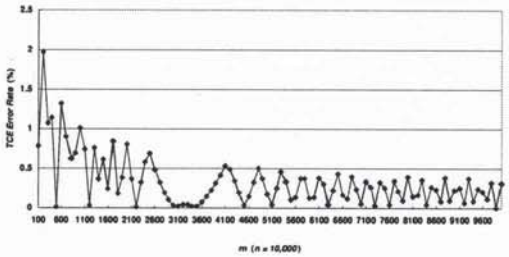


Fig.2: $TCE_{0.01}$ error rate curve of SSSO Algorithm, where n was fixed to 10000 and m was increased from 100 to 10000

In Fig. 3, we fixed the value of m to 10000. We observed that when n was greater than 2000, the accuracy remained almost constant. Thus, we need only let n be greater than 2000 and the value of n will not influence our experiment results.

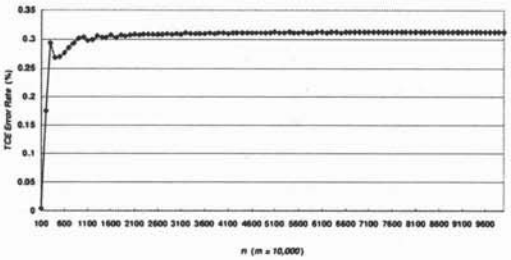


Fig. 3: $TCE_{0.01}$ error rate curve of SSSO Algorithm, where m was fixed to 10000 and n was increased from 100 to 10000

In Fig. 4, we fixed the value of n to 10000. When the value of m approaches 1000000, the accuracy of SSSO Algorithm is higher than 99.95%. When we use SSSO Algorithm to calculate the $TCE_{0.01}$ with m = 1000000 and n = 10000, it takes merely 4.03 seconds for the system to finish calculating. Had we used SSSO-Naive Algorithm to do this, it would have taken almost a half of a day!

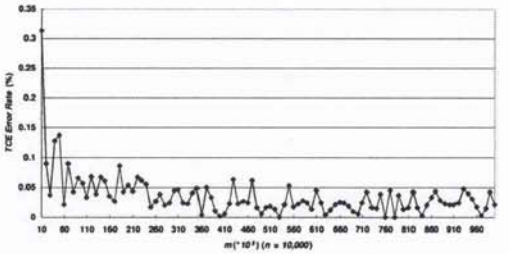


Fig.4: $TCE_{0.01}$ error rate curve, where n was fixed to 10000 and m was increased from 10000 to 1000000

Here, we compared the computation times of SSSO Algorithm and SSSO-Naive Algorithm. In Fig. 5, we fixed the value of n to 10000. We observed that when m increases, the ratio of the computation time of SSSO-Naive Algorithm to the computation time of SSSO Algorithm is also increased. Furthermore, when m is greater than 10000, the ratio is greater than 2500.

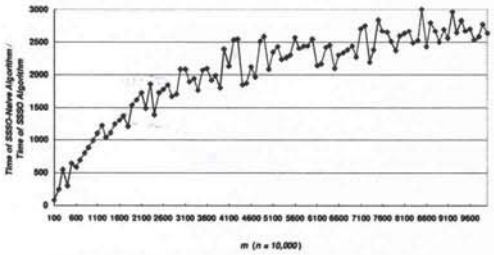


Fig. 5: The ratio of the computation time of SSSO-Naive Algorithm to the computation time of SSSO Algorithm, where n was fixed to 10000 and m was increased from 100 to 10000

In Fig. 6, we fixed the value of m to 10000 and we achieved similar results to those in Fig.5.

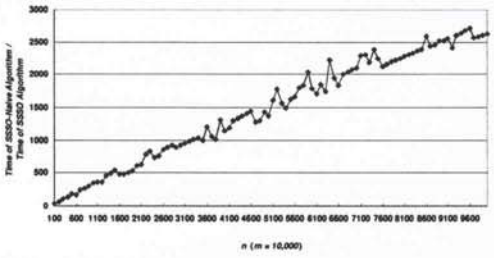


Fig. 6: The ratio of the computation time of SSSO-Naive Algorithm to the computation time of SSSO Algorithm, where m was fixed to 10000 and n was increased from 100 to 10000

Finally, we wanted to know how much time it would take to reach the same accuracy if we were to calculate a quantile smaller than 1%. Hence, in Fig. 7 we used SSSO algorithm to compute $TCE_{0.001}$. The value of $TCE_{p\_benchmark}$ in Equation (17) in this experiment is 4.38054. We fixed the value of n to 10000. We observed that comparing with the results of $TCE_{0.01}$, the value of m slightly increases to reach a certain level of accuracy.

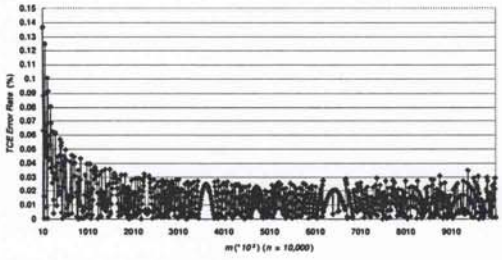


Fig. 7: $TCE_{0.001}$ error rate curve, where n was fixed to 10000 and m was increased from 10000 to 10000000

Fig. 8 shows that when n is fixed at 10000, even if m is equal to ten millions, SSSO Algorithm only takes about 40 seconds to compute $TCE_{0.001}$.

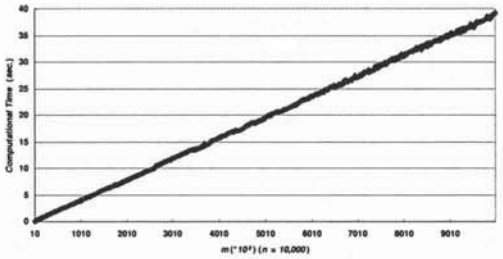


Fig. 8: Computation time of SSSO Algorithm, where n was fixed to 10000 and m was increased from 10000 to 10000000

2) SSMO

In this case, we wanted to compare the computation times for the SSMO-Naive Algorithm, q-times-SSSO Algorithm, and SSMO Algorithm. Because there are three parameters, we fixed two of the parameters to analyze how the remaining parameter influences the computation time. The SSMO-Naive Algorithm takes too much time, so in our experiments we only used ten points to compare our algorithms to the SSMO-Naive Algorithm. In our experiments, we used the algorithm of $O(q*\log(q)+n+m*q+m*q*\log(q)+m*\log(m))$ computational time complexity as SSMO Algorithm because in this algorithm, m is always greater than q.

In Fig. 9, we fixed the value of m to 10000 and q to 10. We observed that the time ratio of the SSMO-Naive Algorithm to our two algorithms increased as n increased from 1000 to 10000.

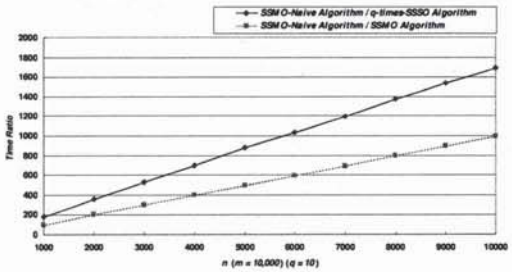


Fig. 9: The ratios of the computation time of SSMO-Naive Algorithm to the computation time of our algorithms, where m was fixed to 10000, q was fixed to 10, and n was increased from 1000 to 10000

In Fig. 10, we fixed the value of n to 10000 and q to 10. We observed that the time ratio of the SSMO-Naive Algorithm to our two algorithms increased as m increased from 1000 to 10000. When m is greater than 6000, the ratios of the time of the SSMO-Naive Algorithm to the time of our algorithms increase very slowly.

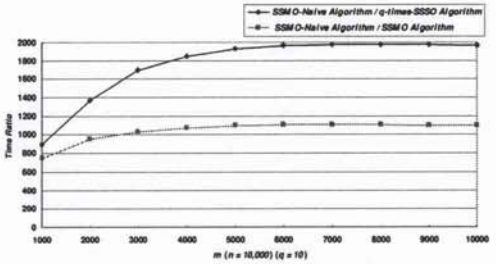


Fig. 10: The ratios of the computation time of the SSMO-Naive Algorithm to the computation time of our algorithms, where n was fixed to 10000, q was fixed to 10, and m was increased from 1000 to 10000

In Fig. 11, we fixed the value of n to 10000 and m to 10000. We observed that the time ratio of the SSMO-Naive Algorithm to our two algorithms increased as q increased from 1 to 10.

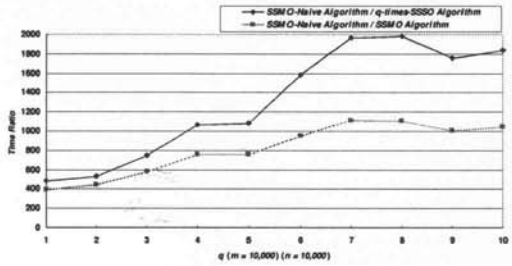


Fig. 11: The ratios of the computation time of SSMO-Naive Algorithm to the computation time of our algorithms, where m was fixed to 10000, n was fixed to 10000, and q increased from 1 to 10

From the experimental results in SSSO, we know that when n is greater than 2000, the value of n does not influence the precision of our results. Furthermore, in this paper, we focus on how much time it takes to get the TCE_p value. We do not want to discuss which kind of portfolio is better. Hence, we will fix the value of both q and n to compare the computation time of q-times-SSSO Algorithm to the computation time of the SSMO Algorithm. In Fig. 12, we fixed the value of n to 10000 and q to 10 and changed m from 100 to 10000 by adding 100 each time. The y-axis represents the time of the SSMO Algorithm minus the time of the q-times-SSSO Algorithm. We observed that in some situations, SSMO Algorithm took less time to compute the results. However, in other situations, the q-times-SSSO Algorithm performed better. We are still not able to confirm under what kind of situation, i.e. different values of m, n, and q, which results perform better though we did observe that when m or q increased from computational time complexity of these two algorithms, the q-times-SSSO Algorithm was more efficient than the SSMO Algorithm. When n increases, the SSMO Algorithm performs better. Therefore, when we handle SSMO, we can use both algorithms at the same time. Moreover, once one of these algorithms finishes calculating the TCE_p value, the results of the slower algorithm may be discarded.

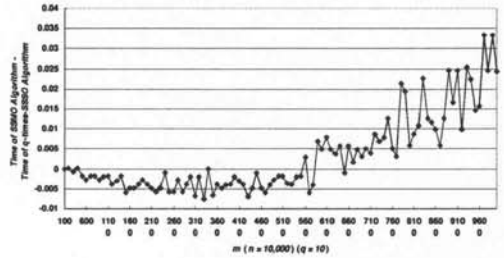


Fig. 12: The difference of the computation time of q-times-SSSO Algorithm to the computation time of the SSMO Algorithm, where n was fixed to 10000, q was fixed to 10, and m increased from 1000 to 10000

VII. Conclusions

In this paper, we have designed efficient algorithms that speed up the computation of the risk measure TCE_p. In the SSSO case, we have reduced the computational time complexity from the SSSO-Naive Algorithm's $O(m*n)$ to the SSSO Algorithm's $O(m+n)$. From the experimental results, we have observed that when we wanted 99.95% accuracy, our algorithms ran thousands of times faster than the SSSO-Naive Algorithm. In the SSMO case, we have provided two algorithms to calculate the TCE_p when there are multiple options in our portfolio. The computational time complexity of the SSMO-Naive Algorithm is $O(q*m*n+ m*\log(m))$. The computational time complexity of the q-times-SSSO Algorithm is $O(q*(m+n)+m*\log(m))$ and the computational time complexity of the SSMO Algorithm is $O(q*\log(q)+n+m*q+ m*q*\log(q)+m*\log(m))$ or $O(q*\log(q)+n+m*q+m*q*\log(m) +m*\log(m))$. Our experimental results also showed that in the SSMO case our algorithms run thousands of times faster than the SSMO-Naive Algorithm.

For future work, we plan to apply our algorithms to several kinds of financial assets, such as convertible bonds, pure bonds, and stocks whose price movements have correlations.

Acknowledgments

We would like to thank Professor Staum at the department of Industrial Engineering and Management Sciences, Northwestern University for providing us with a program to compute the benchmark of the SSSO case in this paper.

References

[1] P. Jorion, Value at Risk: The New Benchmark for Managing Financial Risk, 3rd ed. McGraw-Hill, 2006.
[2] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, "Thinking Coherently," *Risk*, vol. 10, no. 11, pp. 68-71, 1997.
[3] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, "Coherent measures of risk," *Mathematical Finance*, vol. 9, no. 3, pp. 203-228, 1999.
[4] C. Acerbi and D. Tasche, "Expected Shortfall: A Natural Coherent Alternative to Value at Risk," *Economic Notes -Siena-*, vol. 2, pp. 379-388, 2002.
[5] Z. Landsman and E. A. Valdez, "Tail Conditional Expectations for Elliptical Distributions," *North American Actuarial Journal*, vol. 7, no. 4, pp. 55-71, 2003.
[6] Z. Landsman and E. A. Valdez, "Tail Conditional Expectations for Exponential Dispersion Models," *Astin Bulletin*, vol. 35, no. 1, pp. 189-209, 2005.
[7] E. Furman and Z. Landsman, "Risk capital decomposition for a multivariate dependent gamma portfolio," *Insurance Mathematics and Economics*, vol. 37, no. 3, pp. 635-649, 2005.
[8] V. Brazauskas, B. L. Jones, M. L. Puri, and R. Zitikis, "Estimating Conditional Tail Expectation with Actuarial Applications in View," *Journal of Statistical Planning and Inference*, vol. 138, pp. 3590-3604, 2008.
[9] J. Cai and H. Li, "Conditional Tail Expections for Multivariate Phase-Type Distributions," *Journal of Applied Probability*, vol. 42, no. 3, pp. 810-825, 2005.
[10] H. Lan, B. L. Nelson, and J. Staum, "A Confidence Interval for Tail Conditional Expectation via Two-Level Simulation," *The Proceedings of the Winter Simulation Conference*, pp. 949-957, 2007.
[11] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.
[12] J. C. Cox, S. A. Ross, and M. Rubinstein, "Option Pricing: A Simplified Approach," *Journal of Financial Economics*, vol. 7, pp. 229-263. 1979.