Sampling-based Smoothed Analysis for Network Algorithm Evaluation

Xiaoqi Ren*, Zhi Liu*, Yaxuan Qi*†, Jun Li†‡, and Shanghua Teng§
*Department of Automation, Tsinghua University, Beijing, China
†Research Institute of Information Technology, Tsinghua University, Beijing, China
‡ Tsinghua National Lab for Information Science and Technology, Beijing, China
§ Department of Computer Science, University of Southern California, USA
{renxq08, zhi-liu12}@mails.tsinghua.edu.cn, {yaxuan, junl}@tsinghua.edu.cn, shanghua@usc.edu

***Abstract*—Accurate performance evaluation for network algorithms is vital to meet various requirements of different applications, such as QoS, network security, traffic engineering. Although worst-case and average-case analysis are widely used in algorithm evaluation, they are often insufficient due to the lack of practicality. Smoothed Analysis (SA) introduces a new concept of smoothed complexity, remedying the shortcomings in worst-case and average-case analysis. However, recent research towards SA focuses on theoretical evaluation, and thus those methods tend to be too complicated for the analysis of network algorithms. To address the problem, Sampling-based Smoothed Analysis (SSA) for network algorithm evaluation is proposed. SSA extends feasibility for practical performance evaluation and achieves promising experimental results. As examples, two algorithms for typical network problem are evaluated using the proposed SSA framework, and the results explicitly illustrate their significant performance difference in spite of the same theoretical worst-case complexity. Besides evaluation accuracy, SSA also provide more insight for algorithms to facilitate current algorithms improvement and new algorithms design.**

I. Introduction

Performance evaluation of network algorithms is critical to real-life application, e.g. to guarantee Quility-of-Service, to save memory usage, to reduce power consumptions. Accurate performance measurement is particularly vital for algorithms applied to large-scale network applications which receive, process, and respond millions of requests per second with limited computing resources. Processing time and memory usage of the algorithms are strictly limited by realistic demands and capacities of network devices. Thus, objective and reliable performance evaluation plays an indispensable role in network algorithm design and implementation.

Traditionally, worst-case analysis and average-case analysis are widely used in algorithm performance evaluation. However, they are often inadequate for accurate performance measurements due to the following reasons.

1) Worst-case analysis unveils algorithm performance under contrived and extreme circumstances. Thus it may be unduly pessimistic since realistic input datasets may seldom incur worst-case scenarios for algorithms. In practice, some algorithms with high worst-case complexity still achieve great real-life performance [1].

2). Average-case analysis shows average of performance over wholesale inputs. Thus it is hard to obtain average-case performance for those algorithms with unknown input distribution. Even when calculating average-case performance is possible for some algorithms, it tends to result in an optimistic evaluation against practical performance since it eliminates the worse parts and only reflects average feature of all the inputs.

In a nutshell, worst-case or average-case analysis is not always consistent with the practical performance. Smoothed Analysis (SA) was proposed to fill the gap between analytic results and practical algorithm performance by D. A. Spielman and S.-H. Teng in 2004 [1]. SA not only takes the worst case into consideration, but also includes the surrounding area, which leads to higher accuracy for practical algorithm evaluation. A lot of related work endeavor to use SA for evaluating different algorithms, such as 2-Opt Algorithm for the TSP [2], ICP Algorithm [3], Binary Search Trees [4], Integer Programming [5], and Path Trading [6]. However, current research mainly focuses on theoretical Smoothed Analysis for algorithms, which tends to have intrinsic drawbacks to analyze algorithms applied to real-life applications in context of networking for the following reasons.

- The input dimension of network algorithms is high.
- Prior distribution of the whole input space is often unknown.
- Software and hardware limitation is complex and hard to model.

Since mathematical derivation of SA for networking algorithms is infeasible, our approach leverages Sampling-based Smoothed Analysis (SSA) to simplify the process, which utilizes sampling method to overcome the difficulty for mathematical modeling while maintains the benefits of SA.

Our contributions: With a solid underpinning on SA, we propose SSA as a feasible method for practical performance evaluation. To support our framework, we perform worst-case analysis, average-case analysis and SSA, respectively, to evaluate temporal and spatial performance of two algorithms for a typical network problem. While other methods present confusing results, experimental results of SSA is consistent with practical experience, proving that SSA achieves reasonable performance evaluation for network algorithms.

The rest of the paper is organized as follows. In Section 2, we introduce the technical background of this paper; Section

978-1-4799-1353-4/13/$31.00 ©2013 IEEE

3 presents the framework of SSA; as a case study, we evaluate the temporal and spatial performances of two well-known algorithms using SSA in Section 4; in the last section, we state our conclusion.

II. Background

A. Introduction of SA

As is analyzed above, worst-case evaluation and average-case evaluation emphasize different aspects of performance properties and may deviate away from practical performance. In the seminal paper on Smoothed Analysis [1], D. A. Spielman and S.-H. Teng present a novel algorithm evaluation framework to remove the aberration and obain a close-to-real performance analysis. Compared to worst-case analysis and average-case analysis, SA initiatively reflects the density of worst cases. A standard theoretical SA framework is as follows.

P is a problem and A is an algorithm to solve P. We denote $M_A(x)$ as the metric we use to evaluate A on an input x , X as the domain of inputs x for A, σg as a Gaussian random variables with mean 0 and constant deviation σ, $SA(x)$ as the smoothed result for input x, and $C_A^M(x)$ as the complexity for algorithm A under metric $M_A(x)$ on input x. As a general definition, we use E_g to denote the *mean* as g varies randomly. Based on above denotations, the worst-case performance of A is $max(M_A(x))$. The worst-case complexity $C^M_{A_{worst}} = max(C_A^M(x))$. As for SA, a smoothed result of input x is defined as

$$SA(x) = E_g(M_A(x + \sigma g))$$

SA Complexity is defined as

$$max(E_g(C_A^M(x + \sigma g)))$$

A more detailed definition and description can be found in [1] and [7].

Currently, SA is used to analyze many classical algorithms which have high worst-case complexity while perform efficiently in practice. These algorithms include Multi-Level Feedback Algorithm, Integer Linear Programming, Iterative Closest Point (ICP) algorithm, the 2-Opt Algorithm for the TSP.

B. Limitation of Theoretical Smoothed Analysis

In network field, some research uses SA to eliminate low-probability worst instances and obtain close-to-real evaluation results [6]. However, since real-life network problems are intrinsically complicated, theoretical analysis often has to make assumptions and simplify the process and hardware limitations. Therefore, SA loses part of its accuracy and thus becomes an auxiliary method to obtain an upper or lower bound of complexity for an algorithm. In Sec 1, we discuss a general scenario where theoretical analysis cannot receive a satisfactory result. In this part, we will illustrate more about the intrinsic shortages in theoretical analysis via analyzing a typical network problem.

Packet Classification is one of the fundamental network problems and has been widely studied to meet high performance demand of various network applications such as intrusion detection, access control, and flow monitor. Currently, a multitude of algorithms such as RFC [8], HSM [9] and HyperSplit [10] have been developed to address the problem. During the prepossessing stage, these algorithms take multi-dimensional filter set as input and build the corresponding search structures. Afterwards, the algorithms process packet classification via looking up through the structure.

Evaluation of Packet Classification algorithms commonly uses three performance metrics, classification speed, memory usage, and preprocessing time [11]. To conduct performance evaluation based on SA, it is necessary to generalize how realistic inputs varies. A suite of tools for benchmarking named ClassBench [12] reveal the structure properties of the input filter sets. When adjusting the parameters, the data structures will be reconstructed as if some rules are added, deleted or modified in reality, which provides convenience for practical performance evaluation. However, theoretical SA evaluation of Packet Classification algorithms is still a thought-provoking problem and lack of feasibility. Chief difficulty lies in that performance and efficiency of many algorithms are sensitive to the structural and statistical properties of input filter sets. Moreover, traversing all the input cases is infeasible due to the high-dimensional input space and the time-consuming prepossessing stage.

To address the problem of feasibility, we utilize sampling method to improve the original SA and propose SSA. In this paper, we use SSA to evaluate two packet classification algorithms, compare evaluation results with other analysis methods and verify the efficiency and reasonability of our framework.

III. SSA Framework

Based on theoretical SA framework in Sec 2.A, we present a Sampling-based Smoothed Analysis framework (SSA). In SSA, we make several modifications based on theoretical framework to facilitate experimental analysis.

A. Methodology

Input Data Set Generation:

To conduct SSA evaluations while achieve approximate accuracy, we need to generate a large enough input data set for particular input selection corresponding to SA principles.

STEP1: gather N typical cases and constitute a typical-case set S.

STEP2: choose worse cases from typical-case set S and generate a worse-case set W. Here an input x in S is noted as a worse case if and only if $M_A(x) > max_{y \in U(x)}\{M_A(y)\}$. $M_A(x)$ represents the metric we use to evaluate A on an input x. $U(x)$ represents the neighborhood of x in S.

Sampling for Worse-case Set W:

For each x in W, based on Smoothed Analysis principles, we will sample according to a particular distribution in the neighborhood of x.

STEP1: choose joint distribution for sampling on inputs in worse-case set W.

STEP2: sample based on the joint distribution in the neighborhood of each instance x in W, verify the distribution of samples, and generate a sampling set for each x in worse-case set W: $S_x = \{x + \sigma g\}$. The notation of σ and g are the same as Sec 2.A.

Sampling-based Smoothed Analysis for each x in W:

Based on each S_x, we use Smoothed Analysis metrics to evaluate the performance of the algorithm.

STEP1: evaluate performance based on S_x for each x: $M_A(x+\sigma g)$

STEP2: calculate expectations of result set $M_A(x + \sigma g)$ for each x: $E_g(M_A(x + \sigma g))$

STEP3: obtain max of all the expectations
$$max_x(E_g(M_A(x + \sigma g))$$
as SSA result.

B. Modifications and Mathematical Derivation

Theoretical SA is mathematically accurate while hard to generate a general derivation method for all the algorithms. SSA framework modifies theoretical SA to facilitate experimental performance evaluation while maintains the approximate accuracy at the same time. With certain assumptions and enough sampling times, SSA framework is in correspondence with theoretical SA method.

In theoretical method, it is simple to conduct Smooth Analysis for each input in overall input domain and analyze the smoothed performance. However, in experimental method, conducting smoothed analysis experiments for every input in the whole domain is often infeasible. Thus, the original SA formula:

$$max(E_g(M_A(x + \sigma g)))$$

is changed in SSA as:

$$max_x(E_g(M_A(x + \sigma g)))$$

The most significant divergence between SA and SSA is smoothing each input in the whole input plane or just smoothing inputs in a particular set. When we choose enough cases into the particular sampling set, the two approach converge to a uniform result. An extreme example is that when the number of sampling cases in the sampling set is infinite, the set of sampling cases approximately covers the overall input plane. Then, the two approaches will generate the same results.

Meanwhile, it is reasonable to choose part of inputs, for example worse cases, as sampling cases to represent other inputs if performance metric for inputs is mathematically smoothed enough. Because in this case, we can expect performances of inputs around worse cases are also not so good so that the worst sampling result for worse cases can represent the worst sampling result for the whole input plane. Thus, the experimental Smoothed Analysis result may be nearly the same as the theoretical SA analysis.

C. Discussion of the Framework

In this part, we will discuss some bewildering elements in designing and conducting real experiments under the guidance of SSA. Our discussion mainly focuses on two aspects, *"Worse Case" Generation* and *Sampling Frequency*. We hope our discussion can inspire further study and research to improve the accuracy and feasibility of SSA method.

***"Worse Cases"* Generation** Selection of "Worse Cases" is a critical step for SSA. As is defined in Section 3.A, "Worse Cases" are a set of cases which satisfied the formula:

$$M_A(x) > max_{y \in U(x)}\{M_A(y)\}$$

"Worse Cases" generation need not traverse all the input plane, which is often impractical. In other words, "Worse Cases" are local maximums on input plane for performance metric. Numerical analysis provides us with several efficient methods to find local maximums from the whole input plane [13].

Sampling Frequency Sampling frequency significantly influences the accuracy of experimental results. Higher sampling frequency will contribute to more accordance with theoretical SA. Meanwhile, higher sampling frequency will also enlarge experimental difficulty since it extends the time period for conducting SSA, especially for those algorithms which have a long processing time. Thus, to obtain a proper sampling frequency is to trade off between the correctness and feasibility.

When the sampling frequency increases, the experimental results will approximate to theoretical analysis. We can use the variation of evaluation results as a measurement to select a proper frequency. If the variation of performance evaluation results is smaller enough, e.g. smaller than a threshold, when sampling frequency increases, it is believed that the current sampling frequency does well enough for the performance metric.

IV. Case Study

In this section, we use SSA to evaluate the performance of two algorithms for a typical network problem, Packet Classification problem. We will show that SSA is more capable of revealing the practical performance of network algorithms.

For confidential concerns, it is impossible to get enough real rule filters as typical inputs. In all the following experiments, we utilize ClassBench [12] to generate inputs and to obtain samples in the neighborhood of a given worse case. ClassBench generates synthetic while realistic filters based on a filter seed and provides three controlling parameters naming *smoothing*, *port scope*, and *address scope*. We vary the three parameters in their own range, respectively, and consider the generated filter sets as the inputs for the following algorithms.

A. Algorithm Introduction

We utilize two classical packet classification algorithms as case study, Computational Geometry Algorithm [14] and HyperSplit Algorithm [10].

Computational Geometry Algorithm Computational Geometry Algorithm uses a basic KD-Tree data structure. It chooses

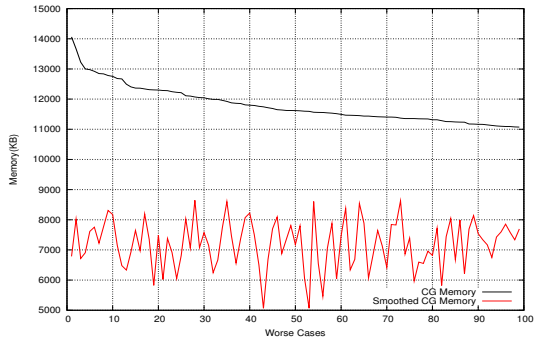


Fig. 1: (CG) Worse Case Memory Size before and after SA

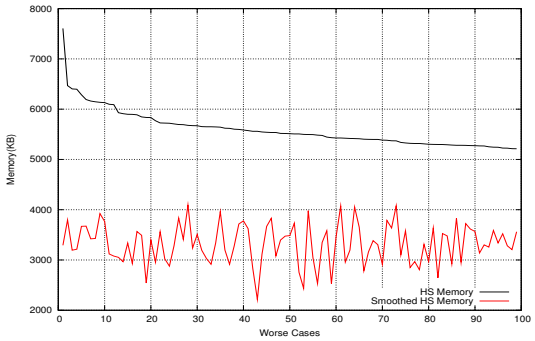


Fig. 2: (HS) Worse Case Memory Size before and after SA

a dimension to split the high-dimensional space in order, that is, from dimension 1, dimension 2, ... , to dimension n, then restarting from dimension 1, over and over again until the space is split small enough. However, the memory occupancy of Computational Geometry Algorithm is always high due to the overlaps of the rules, which leads to considerable overhead of memory accessing. Therefore, Computational Geometry Algorithm hardly meets the requirement of high-performance empirically.

HyperSplit Algorithm HyperSplit algorithm uses an optimized KD-Tree data structure for packet classification. It combines the advantages of both parallel search and tree search algorithms by using a simple but efficient binary search tree for classification. Unlike the basic KD-Tree data structure, HyperSplit uses rule-based heuristics to select the most efficient splitting point on a specific field. Compared with Computational Geometry Algorithm, HyperSplit consumes much less memory with little sacrifice in terms of tree depth. It is believed that HyperSplit is able to utilize higher cache hit rate and achieve practicability under most realistic circumstance.

TABLE I: Worst and Average Memory Results

Algorithm	Worst-case Memory(KB)	Average Memory(KB)
CG	14061	1769.01
HS	7606	763.24

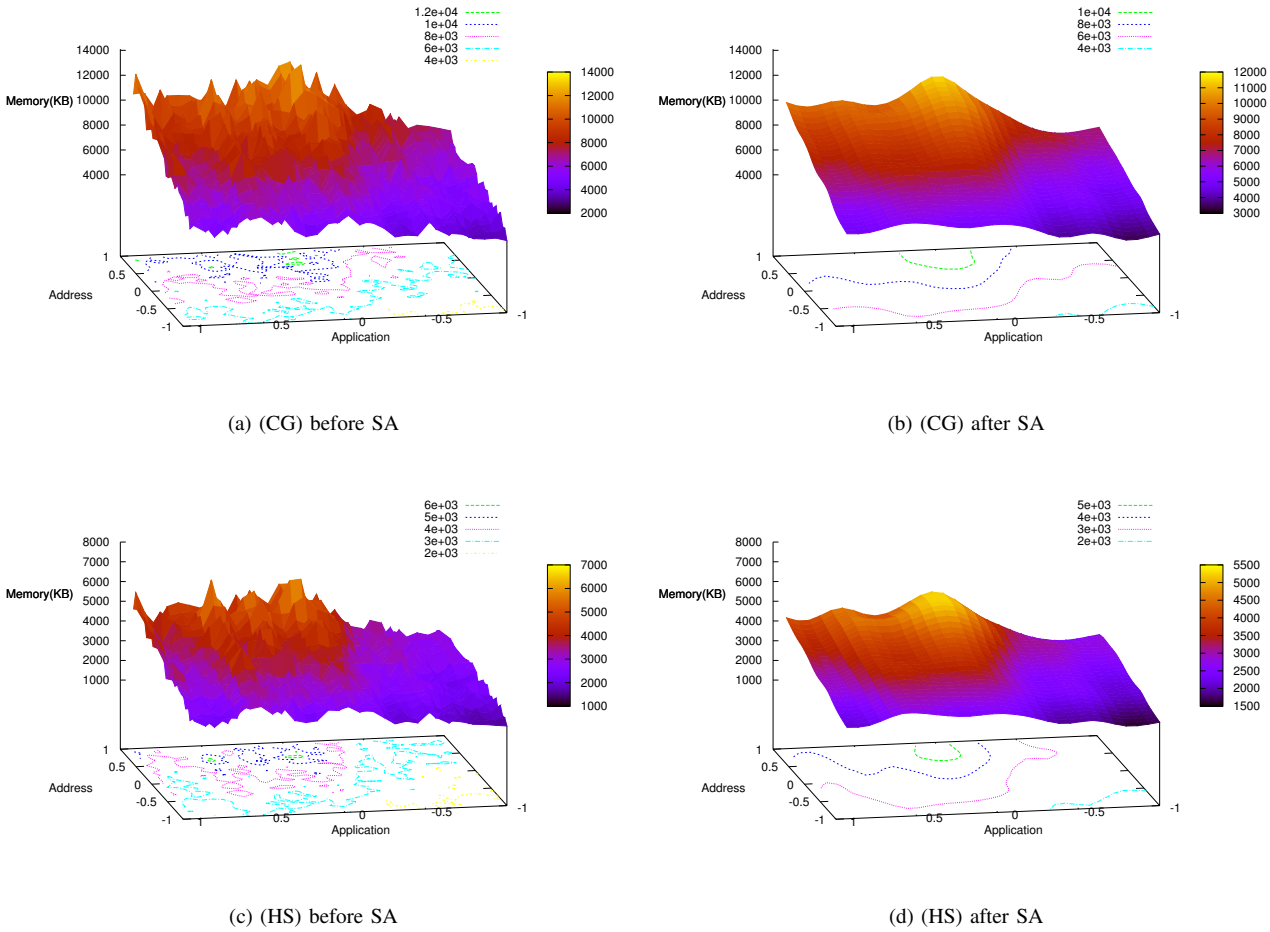
B. Evaluation of Spatial Performance

As is mentioned in Sec 2.B, memory size is one of the key metrics in packet classification algorithms evaluations. The average-case and worst-case memory size for Computational Geometry Algorithm (CG) and HyperSplit Algorithm (HS) is showed in Table I . From the table, worst-case memory usage of the two algorithms are large, which indicates that neither of them is adequate for practical use. Meanwhile, their average-case memory usage are small. As a result, worst-case analysis and average-case analysis present totally different evaluation results for the performance and are inconsistent with our experiences. To deal with the conflict, we utilize SSA to obtain a further insight into the practical performance of the two algorithm. Later we will show that the experiment results are consistent with our practical experience and that HS outperforms CG in terms of memory usage.

Using SSA framework, for CG and HS, we choose the first 100 worse cases out of more than one hundred thousand input sets and conduct SSA experiments under the guidance of our framework, respectively. Figure 1 depicts the memory occupancy for Computational Geometry Algorithm before and after Smoothed Analysis, and Figure 2 illustrates the memory usage for HyperSplit Algorithm before and after Smoothed Analysis. Note that, the SSA curves in Figure 1 and Figure 2 are *"jagged"* rather than *"smoothed"*. The surrounding of a worse case may be a "peak" or a "plateau". Here a "peak" represents the situation that memory sizes of the surrounding points decrease rapidly from the center point, namely, the worse case we chose. A "plateau" depicts the situation that memory sizes of the surrounding points decrease slightly and slowly from the center point. The SSA result should be smoothed if we choose consecutive points. However, in this case, since our worse cases scatter randomly on the input plane and may be "peak-like" points or "plateau-like" points, it is naturally that the curves depicted the memory size after SA are jagged rather than smoothed.

From Figure 1 and Figure 2, it is obviously that after Smoothed Analysis, the memory size decreases to almost half of the value of the peak point for both algorithms. The result curves illustrated that the two algorithms are stable and only have several peak points which result in high memory usage. These results clearly show that the memory occupancy of HyperSplit after SSA is small enough for the cache in a large number of current processors, while CG is more likely to exceed the cache volume and result in higher cache miss rate. Thus, our SSA framework explains why HyperSplit is feasible and efficient in practice while has a theoretical exponential worst-case complexity of $\Theta(N^F)$, where N is the non-overlapping hyper-rectangles generated from filter set and F is the number of fields in packet header [10].

To show how SSA works more clearly, we plot panoramas to depict how memory usage varies while parameters vary. Since we have three parameters to control the input filters, we fix the *smoothing* parameter and conduct SSA based on the other two parameters to build a three-dimensional



(a) (CG) before SA

(b) (CG) after SA

(c) (HS) before SA

(d) (HS) after SA

Fig. 3: Memory Size before and after Smoothed Analysis

figure. Figure 3a and Figure 3b show how memory usage varies for CG algorithm before and after Smoothed Analysis, respectively. Similarly, Figure 3c and Figure 3d illustrate how memory usage varies for HS algorithm before and after SSA. Also, we depict the contour lines for different memory size in each figure to illustrate the fluctuation of memory size. For both algorithms, almost all peaks vanish after SSA, which means the distribution of worse cases is disperse. In other words, it is hard to be entrapped into a worse case plateau for real filter sets.

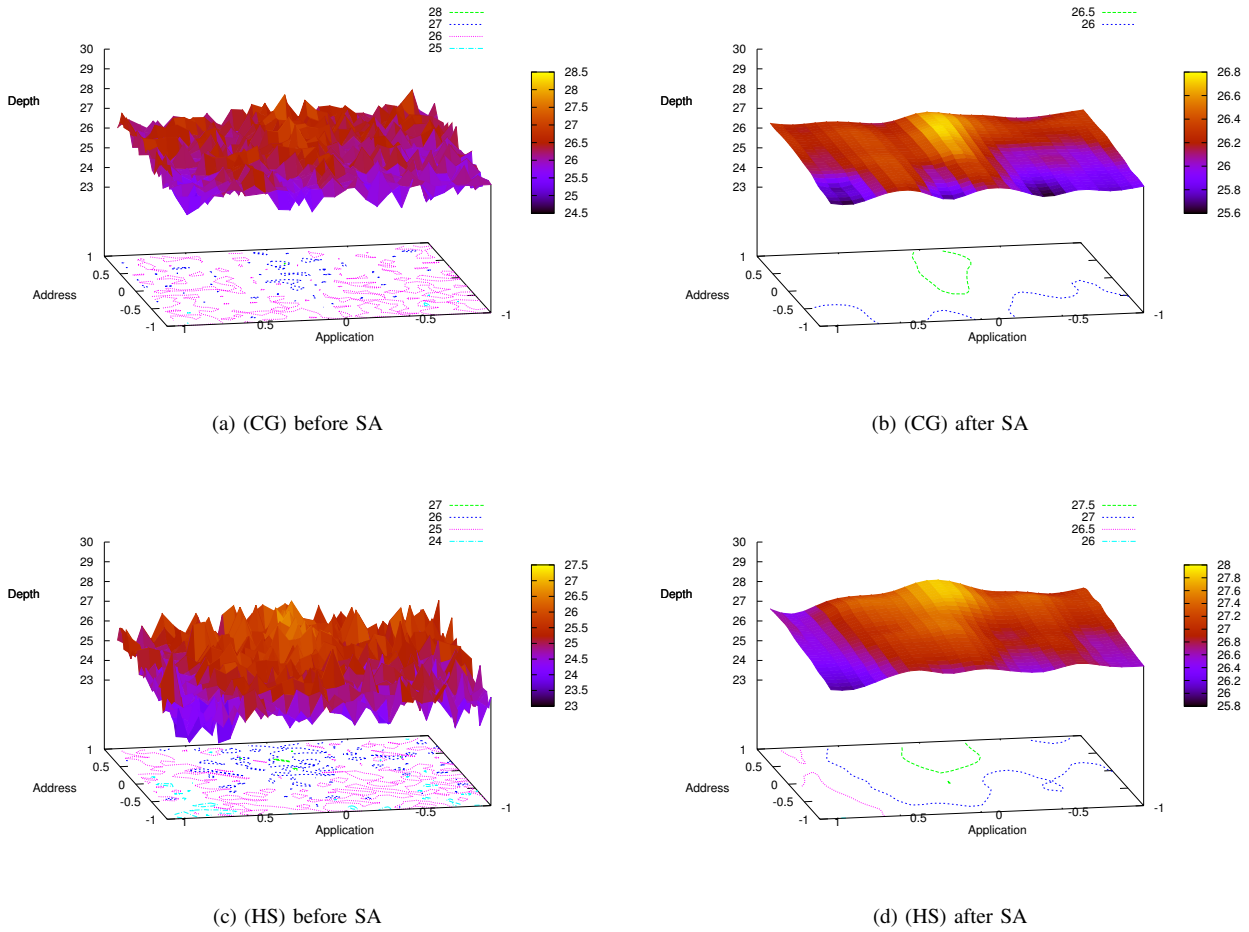
C. Evaluation of Processing Speed

SSA also provides more insight into practical performance comparison of network algorithms. Mostly, the input sets of network algorithms are not constant and often come up with some minor modifications, e.g. adding rules, deleting rules or changing the ranges of rules. Thus, users tend to care more about the performance within certain input areas rather than the performance at each input point. In this part, we varies two parameters of ClassBench to evaluate on the depth of search trees, which represents the possible worst search times and is considered a significant measurement to evaluate processing speed [10]. Figure 4 depicts how depth varies before and after SSA. Figure 4a and Figure 4b shows varying of worst depth for CG algorithm before and after SSA. Likewise, Figure 4c and Figure 4d illustrate the distinctions for HS algorithm. Based on the observation of the contour line in Figure 4b and Figure 4d, we can find that CG narrowly wins HS on the metric of depth within the corresponding input areas. However, when comparing Figure 4a and Figure 4c before SSA, it is difficult to tell which algorithm has a better performance.

V. Conclusions and Future Work

In this paper, we propose a framework named SSA for network algorithm performance evaluation. The framework originates from SA and achieves feasibility and reasonability of practical performance evaluation for network algorithm with complex processing and high-dimensional input space. The experiment results on two well-known packet classification algorithms draws the preliminary conclusion that, while in some situations worst-case analysis and average-case analysis may have conflict or unclear results, SSA is able to reveal the practical performance.

However, it is not theoretically proved that how SSA will converge to SA as the sampling rate increases, which will be conducted in our future work. Besides, we will apply SSA



(a) (CG) before SA

(b) (CG) after SA

(c) (HS) before SA

(d) (HS) after SA

Fig. 4: Worst Depth before and after Smoothed Analysis

to other typical network algorithms, obtaining more insight into algorithm comparison and the improvement of SSA framework. Study on the method of worse-case generation is also needed to help SSA achieve higher efficiency. As to sampling models and sampling rate, we will try distributions other than Gaussian randomness and determine the sampling rate by means of input space analysis.

VI. Acknowledgement

The authors would like to thank Jialing Zhang and Baohua Yang for their contributions. This work was supported by Tsinghua National Laboratory for Information Science and Technology.

References

[1] D. Spielman and S. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *Journal of the ACM (JACM)*, vol. 51, no. 3, pp. 385–463, 2004.

[2] M. Englert, H. Röglin, and B. Vöcking, "Worst case and probabilistic analysis of the 2-opt algorithm for the tsp," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1295–1304.

[3] D. Arthur and S. Vassilvitskii, "Worst-case and smoothed analysis of the icp algorithm, with an application to the k-means method," in *FOCS'06*.

[4] B. Manthey and R. Reischuk, "Smoothed analysis of binary search trees," *Theoretical Computer Science*, vol. 378, no. 3, pp. 292–315, 2007.

[5] H. Röglin and B. Vöcking, "Smoothed analysis of integer programming," *Integer Programming and Combinatorial Optimization*, pp. 276–290, 2005.

[6] A. Berger, H. Röglin, and R. Van Der Zwaan, "Path trading: fast algorithms, smoothed analysis, and hardness results," *Experimental Algorithms*, pp. 43–53, 2011.

[7] D. Spielman and S. Teng, "Smoothed analysis: an attempt to explain the behavior of algorithms in practice," *Communications of the ACM*, vol. 52, no. 10, pp. 76–84, 2009.

[8] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proc. ACM SIGCOMM*, 1999.

[9] D. Xu, B. Jiang and L. J., "Hsm: A fast packet classification algorithm," in *Proc. of the 19th Advanced Information Networking and Applications (AINA)*, 2005.

[10] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: from theory to practice," in *INFOCOM 2009, 28th Annual IEEE International Conference on Computer Communications.*

[11] A. Feldman and S. Muthukrishnan, "Tradeoffs for packet classification," in *INFOCOM 2000. 19th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, pp. 1193–1202.

[12] D. Taylor and J. Turner, "Classbench: A packet classification benchmark," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, pp. 2068–2079.

[13] C. Gerald and C. Green, *Numerical analysis*. Addison, 2003.

[14] M. De Berg, O. Cheong, and M. Van Kreveld, *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc, 2008.