HCF: A Starvation-free Practical Algorithm for Maximizing Throughput in Input-Queued Switches

John Kim Abhishek Das
Computer Systems Laboratory
Stanford University, Stanford, CA 94305, USA
{jjk12, abhishek@cva.stanford.edu}

Abstract—
Using virtual output queueing(VOQ), maximum matching scheduling algorithms have been shown to achieve 100% throughput in input-queued switches, but has high complexity such that implementation is infeasible for high-speed systems. Iterative maximal matching algorithms, proposed as an alternative, cannot run for more than a few iterations due to the hardware complexity involved, thus resulting in low throughput. In this paper, we introduce a starvation-free iterative maximal matching algorithm called Highest Count First (iHCF). The iHCF algorithm gives preferential service based on the approximate age of the head-of-line cell in a VOQ and maximizes the size of the matching using round-robin priority pointers. We show that iHCF can achieve 100% throughput under i.i.d and uniform traffic in a single iteration. We also show using simulations that it performs as well as other known practical algorithms and achieves 100% throughput when run for only a few iterations under different admissible traffic patterns. Compared to other algorithms, iHCF leads to a low complexity architecture such that the scheduling does not become the bottleneck.

I. Introduction

The exponential growth of multimedia applications and the popularity of World Wide Web has dramatically increased the amount of traffic carried over today's high-speed backbone networks. As a result, there is great demand for high-speed, high-aggregate bandwidth switches and routers. A high performance switch is needed to take data arriving on an input link and quickly deliver it to the appropriate output link. Output-Queued (OQ) switches are known to provide an optimal delay-throughput performance and even quality-of-service(QOS)[1], for all traffic distributions, but requires the buffer memories to run N[1] times faster than the line rate. Input-Queued(IQ) switches replace the centralized shared memory by maintaining separate queues at each input of a crossbar fabric, each of which runs at the line rate. However, head-of-line (HOL) blocking is well known to limit the throughput of input-queued switches to approximately 58.6% [2], under the most benign conditions: i.i.d and uniform traffic. This problem can be overcome by using Virtual Output Queues (VOQs) but requires a scheduling algorithm to schedule cell transmission across the switching fabric. It has been shown that a scheduling algorithm that uses a maximum size bipartite matching algorithm can increase the throughput from 58.6% to 100% when the traffic is uniform and independent. However, for non-uniform traffic, maximum size matching has its limitation[5]. Maximum *weight* matching algorithms have been shown to achieve 100% throughput under non-uniform traffic for independent arrivals[5][8]. However maximum size and maximum weight matching algorithms are complex to implement and iterative matching algorithms are often used in practice.

[1] N is the number of switch ports

In this paper, we propose a new iterative weighted matching algorithm iHCF (Highest Count First) which uses weights based on the waiting time of the HOL cells. The weights are approximated by having counters at each VOQ. The algorithm provides high throughput and performs well on a wide range of traffic patterns in only a few iterations. We also present a low complexity implementation, similar to iSLIP, which leads to faster scheduling.

The paper is organized as follows: In Section II, we discuss other known matching algorithms used for scheduling IQ switches. We describe iHCF and its properties in Section III. In Section IV, we compare it to other practical algorithms, both in terms of hardware complexity and throughput performance. Finally, we provide a variation to the algorithm in Section V which overcomes some of the limitations and we conclude in Section VI.

II. Related Work

Most practical algorithms that are fast and simple to implement in hardware approximate a maximum size matching algorithm by a maximal size matching, e.g. PIM [3], iSLIP [4] and DRRM [9]. These algorithms perform well on uniform traffic, but in reality traffic is not uniform. Queues with heavy traffic can overflow while ones with light traffic are empty most of the time and limit the throughput. A maximum size matching algorithm does not consider queue lengths, and hence cannot prevent queue overflow.

It has been proven that by using a maximum weight matching algorithm, 100% throughput can be reached for independent arrivals: Longest Queue First (LQF)[5] and Oldest Cell First (OCF)[8]. The weight of the edge $w_{i,j}$, from input i to output j is a measure of the level of congestion, e.g. the length of queue or the age of its oldest packet. However, implementing these maximum matching algorithms is infeasible for high-speed systems due to the large number of multi-bit comparators required [4] [8], and its $O(N^3 \log N)$ run-time complexity (scheduling delay). Recently, the Longest Port First(LPF)[7] algorithm was proposed to overcome the complexity of LQF, which removes the comparators from the critical path. However for practical implementations, heuristic approximations of maximum size matching (e.g. iSLIP) are often used[8]. In fact, iterative maximal matching algorithms have been proposed as a practical alternative to the above-mentioned algorithms[7][8]. When run to completion, they produce a maximal matching. However even in these iterative alternatives, the complexity of the implementation limits the cycle time (i.e. the time to run 1 iteration) and

0-7803-8924-7/05/$20.00 (c)2005 IEEE.

repeat

Request phase:

1. Each unmatched VOQ with a HOL cell sends request to its output

2. Corresponding VOQ counter increments by 1

Grant phase:

1. Each unmatched output grants the input requesting with the highest counter value

2. Ties are broken *randomly* or based on *priority*

Accept phase:

1. Each input accepts the output grant with the highest corresponding VOQ counter value

2. Ties are broken *randomly* or based on *priority*

3. Each matched VOQ has its counter resetted

until $\exists Q_{ij}$ with HOL cell where i and j are both unmatched

Fig. 1. The three-phase iHCF algorithm with multiple iterations

hence the number of iterations. Thus, it becomes costly to allow the mathcing algorithms to run to completion.

III. The *i*HCF Algorithm

A. Algorithm Overview

The iHCF algorithm has been designed to be an approximation to iOCF but with complexity similar to iSLIP. Like other iterative algorithms, it is composed of three phases: *Request*, *Grant* and *Accept*, as illustrated in Figure 1. Unlike iOCF, iHCF maintain weights by tracking the waiting time of only the HOL cells in each VOQ. It does so by using a counter, voq_cntr_{ij}, to count the number of times each VOQ, Q_{ij}, makes a request from input i to output j. Grants for an output j are made by giving preferential service to inputs with the highest voq_cntr_{ij} value. Similarly, an input i accepts a grant and forms a matching by giving preferential service based on the counter values of the VOQs with grants. Ties in either phase can be broken randomly or in a round-robin manner. Once accepted in a matching, voq_cntr_{ij} is reset to 0. Since the voq_cntr_{ij}s are reset on getting served, each voq_cntr_{ij} eventually becomes the highest count. Thus every VOQ Q_{ij} will be eventually matched and the algorithm will prevent starvation.

By considering only unmatched inputs and outputs, each iteration only increases the size of the matching. Thus iHCF converges to a maximal matching in at most N iterations, which is an approximation to the maximal weight matching produced by OCF. An example of the algorithm is shown in Figure 2.

B. Ties

The size of a matching generated by an iterative matching algorithm is constrained by ties/collisions at outputs and inputs, during the grant and accept phases respectively. Since iHCF grants and accepts on a preferential basis, ties can only occur when there are multiple VOQs with their corresponding voq_cntr_{ij}s having the same highest count. Amongst the contending VOQs for an output j only one VOQ Q_{ij} is granted, whose voq_cntr_{ij} is *reset* when accepted for a match. Moreover, it is the only counter at input i that gets *reset*. This makes

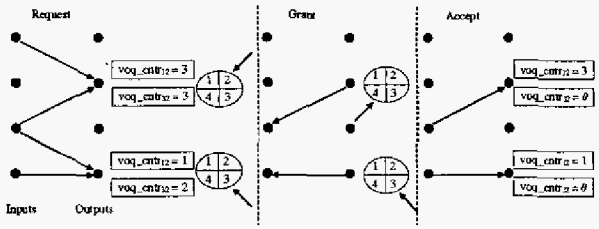


Fig. 2. Example of iHCF algorithm. Similar to the iSLIP algorithm, the priority pointers are only updated when the grant is accepted. is accepted. For the iHCF, there is an additional requirement that the priority pointer does *not* get updated if there were no ties, i.e. if the priority pointer is not used to determine the winner.

the voq_cntr_{ij} *de-synchronized* from other contending counters at input i and output j, reducing the chance to clash again. Thus, ties can only occur when new cells arrive into *empty* VOQs, which is rare under heavily loaded traffic[2].

When ties do occur, ties can be broken in different ways. Randomly breaking ties, as in PIM [3], could lead to unfair bandwidth allocation [6] and are an order of magnitude more complex to implement than iSLIP. We show in Section IV that using randomness to break ties result in poor performance for iHCF. The priority pointers, like in iSLIP, help to desynchronize the input and output arbiters into a round-robin schedule, thus preventing any VOQ from getting starved and converging quickly to a maximal match and achieving higher throughput. We borrow this idea of pointer slipping for iHCF, by keeping a pointer at each input and output with rotational priority to break ties:

1: ties are broken in favor of the one that appears next in a fixed, round-robin schedule starting from the highest priority element;

2: input pointers are incremented (modulo N) to one location beyond the accepted output, *only in case of ties*;

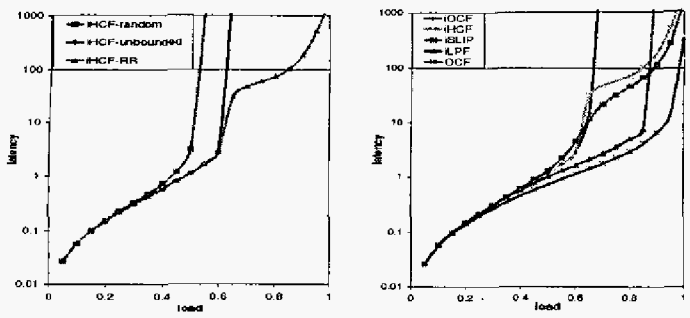
3: output pointers are incremented (modulo N) to one location beyond the granted input, *only if the grant was accepted.*

C. iHCF Counters

A main component in the iHCF algorithm are the counters: voq_cntr_{ij}s. In order for iHCF to be a practical algorithm, the voq_cntr_{ij}s can only be allowed to count to a certain maximum value(MAX_CNT). The size of the counter, i.e. number of bits, in-turn determines the size of the comparator in the input and output arbiters.

With iHCF, the maximum HOL waiting time for a cell is N^2 slots since in the worst case, due to ties, only a single edge is matched in every iteration. Since the counter for an unmatched edge increments every time-slot, it will become the highest count in N^2 time-slots while each of the other VOQs are served and their corresponding counters reset. As a result, the voq_cntr_{ij}s need to count to a maximum value of N^2, and hence require at most $2log(N)$ bits. However, as later shown in Section IV, optimal results are obtained when the counters are only N bits wide.

[2]When voq_cntr_{ij} is bounded, ties can also occur when the counters reach the maximum value.



(a) (b)

Fig. 3. Performance Comparison on Uniform Random Traffic for Switch Size of 16 with 1 iteration (a) iHCF algorithm and its variants (b) iHCF comparison to other algorithms

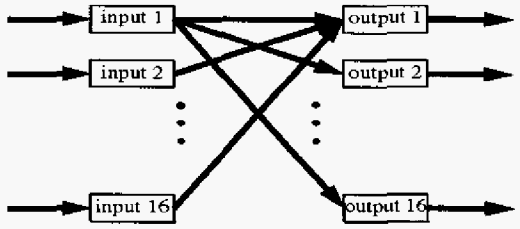


Fig. 4. Non-uniform Traffic Pattern where input 1 and output 1 are the hot spot. Every input sends traffic to output 1 and input 1 sends traffic to every output

If the counters are 1-bit wide, every non-empty VOQ counters will be tied and hence, it is easy to observe that iHCF will behave exactly like iSLIP. Using the iHCF algorithm with a saturating counter, whose MAX_CNT is less than N, 100% throughput can also be achieved for uniform random traffic. Due to space constraint, the proof is not shown but it also approaches the behavior of iSLIP at high loads.

IV. Comparison

A. Performance Results

The following simulations were done using SIM[10] and modified to implement our algorithms. We compare the algorithms for only a single iteration unless otherwise noted. Note that iterative maximal matching algorithms may not run to completion in a single iteration.

A.1 *iHCF algorithm performance*

We first compare the performance of iHCF (1 iteration) and its following variants under uniform traffic load:

1. with unbounded counters
2. with saturating counters and random breaking of ties
3. with saturating counters and round-robin tie breaking

For the saturating counters, the counters were set to $log(N)$ bits. The results are compared in Figure 3(a). iHCF using unbounded counters does not provide 100% throughput, while using saturating counters and breaking ties in a round-robin manner does. Breaking ties randomly does not provide 100% throughput due to the same reason as PIM[3] and provides limited throughput in a single iteration. All subsequent simulation of iHCF refers to the third implementation which uses saturating counters and breaks ties in a round-robin manner.

A.2 *Uniform Traffic*

To evaluate the performance of iHCF, it is compared to the following practical algorithms: iOCF, iSLIP, and iLPF. The OCF algorithm is also shown as it uses maximum weight matching algorithm scheduling algorithm and provides a theoretical bound on an algorithm we are trying to approximate.

The performance under uniform random traffic is shown in Figure 3(b). As expected, the iSLIP algorithm performs well but other iterative algorithms such as iOCF and iLPF do *not* provide good throughput with only a single iteration. iHCF, on the other hand, provides 100% throughput and lower latency at intermediate loads (between 0.4 and 0.6), but higher latency at higher loads compared to iSLIP. At the intermediate loads, iHCF provides better latency because it takes into account the weights approximated by the counters. At higher loads, it has longer latency because it takes longer to desynchronize the counters with the help of the round-robin pointers; where as in iSLIP, only the round-robin pointers need to desynchronized to behave like time division multiplexing at heavy loads.

A.3 *Non-uniform Traffic*

Beside the uniform traffic pattern, we simulate two non-uniform traffic patterns. The first non-uniform traffic pattern is a synthetic workload used in [8], a traffic pattern that is known to perform poorly with iSLIP:

$$\Lambda = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \end{bmatrix}$$

The results of this traffic pattern are shown in Figure 5(a). The iSLIP algorithm only provides a throughput of approximately 70% and even iOCF performs poorly. Both iLPF and iHCF perform well and give 100% throughput.

Even though the above traffic pattern is non-uniform, the total traffic to each input and each output are identical. To test the algorithm with a different traffic pattern, a non-uniform traffic pattern shown in Figure 4 is used which represents a hot-spot traffic [8], e.g. input 1 and output 1 represent the hot spots which have more traffic than the other input and outputs. Because iHCF only considers the HOL cells, we expect this traffic pattern to be one of the adversarial traffic patterns since only a few VOQs are stressed.

As shown in Figure 5(b), although iHCF does provide 100% throughput, it results in higher latency compared to iOCF and iLPF. This is mainly because iHCF does not consider the length of the queue but just the HOL waiting time. However, because it does consider HOL waiting time, it still provides better latency than iSLIP.

B. Complexity Comparison

As stated earlier, iHCF is a heuristic for iOCF and the block diagram for its implementation is very similar to that of iOCF which is described in [8]. However, some of its complexity advantages over iOCF are:

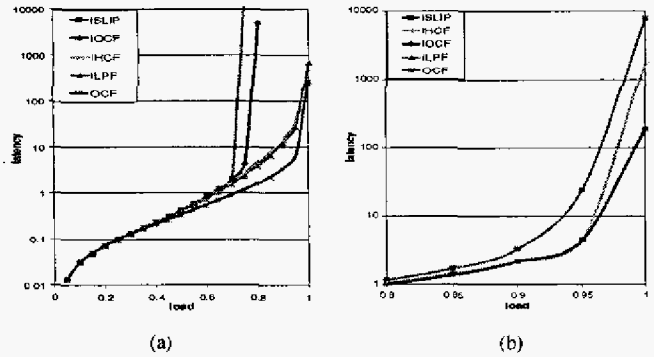


Fig. 5. Performance comparison on (a) non-uniform traffic pattern and (b) hot-spot non-uniform traffic pattern of figure 4

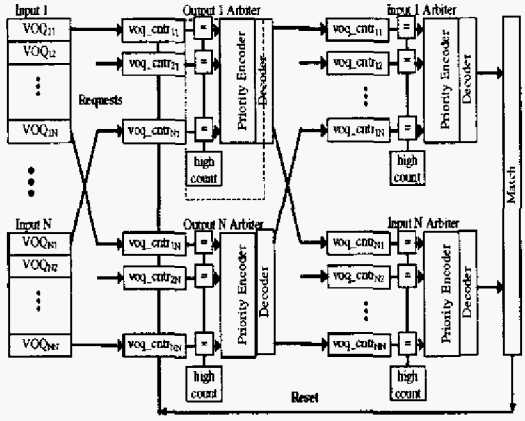


Fig. 6. Block Diagram of iHCF algorithm to reduce critical path. The voq_cntr_{ij} are duplicated for illustration purpose. The outlines box shows the logic which replaces the comparator on the critical path. The comparators, not shown in the figure, are used to calculate the highest two values such that if the input is granted and there were no ties, the second highest value will be used as $high\_count$ in the subsequent cycle.

1. The width of the comparators is significantly reduced since weight is the size of the counter, which is approximately $log(N)$ instead of $log(L)$ where N is the number of switch ports and L is the length of the queues.
2. Each input only needs to send a single bit request instead of $log(L)$ bits which is needed in iOCF.
3. The weight of only the HOL cell needs to be maintained instead of the weight of all the cells in the queue.

In spite of these simplifications, the overall benefit is still minimal as the critical component, the comparators which was shown to consume approximately 88% of the delay [8] still remain on the critical path. Although our algorithm has reduced some of the complexity, the depth of the comparators, which is proportional to the switch size, has not changed. Another problem with this implementation is that ties among the weights are broken randomly which has been shown earlier to result in poor performance.

To overcome the complexity challenges mentioned above, we present a new implementation shown in Figure 6. This implementation removes the comparator from the critical path and ties are broken in a round-robin fashion, similar to the iSLIP algorithm with a programmable priority encoder. This block diagram resembles the iSLIP implementation which is known to make fast scheduling decisions. Instead of the requests feeding directly to an iSLIP arbiter, the request is fed to the arbiter only if the weight(voq_cntr_{ij}) matches the highest value amongst all the voq_cntr_{ij}'s for a given input i or a given output j: $high\_count$. If there is only one voq_cntr_{ij} that matches the highest value (i.e. there are no ties), the iSLIP arbiter can be bypassed. Thus, the only delay added to the iSLIP hardware implementation is the delay through one counter and two single stage comparators. From our estimations, this new logic will add less than 10% to the iSLIP delay, which is significantly smaller than the scheduling time required for iOCF or iLPF.

This implementation takes advantage of the fact that the highest counter value, $high\_count$, used in the comparators can be predicted by making the following observations:

- if $high\_count$ is MAX_CNT, but is not granted, it remains at MAX_CNT
- if there was a tie OR if the output was not granted, then $high\_count = high\_count + 1$
- if there was no tie and the output was granted, the predicted value of $high\_count$ will be the previous high value incremented by 1.

To determine the previous highest value, a comparator is still needed, but it is no longer part of the critical path and has no impact on the delay of the scheduling. The same technique used in iHCF can not be applied to the iOCF algorithm since after the VOQ is served, the weight of the next cell cannot be predicted.

V. iHCF Variant for Unbalanced Non-uniform Traffic

In section IV-A.3, we considered non-uniform traffic patterns which balances the load across different inputs. In this section, we show how iHCF does not perform well on traffic patterns where the load is not balanced across the inputs. As an example, we modify the traffic pattern shown in Figure 4 such that each input has a *hot* output, with a fraction p of cells from an input destined to its *hot* output, and other fraction of cells $(1-p)$ uniformly destined to other outputs. This is the same traffic pattern used in [9]. As shown in Figure 7, iHCF with even 4 iterations performs poorly.

iHCF is an attempt to approximate the waiting time of a cell. However, it only considers the HoL waiting time(W_{HoL}), whereas the total waiting time (W_{total}) is

$$W_{total} = W_{HoL} + W_{queuing}$$

The waiting time behind other cells ($W_{queuing}$) can be approximated by the queue length of the VOQ when the cell reaches the head of the queue(L_{queue}).[3] Since at most one new cell arrives, and hence increases the queue length, during each time slot the cell waits behind other cells,

$$L_{queue} \leq W_{queuing}$$

[3] L_{queue} is different from L which represents the total queue length. L_{queue} represents the instantaneous queue length when the cell reaches the head of the queue

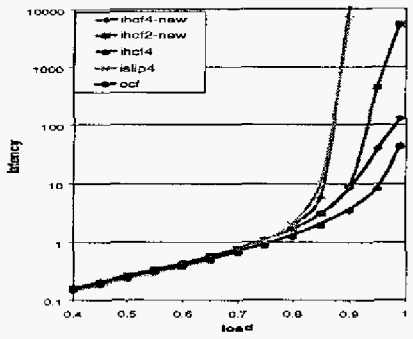


Fig. 7. iHCF algorithm variant with counters for queue length. Hot spot traffic with p = 0.8 is used.

Thus, the total delay can be approximated as

$$W_{total} \approx W_{HoL} + L_{queue}$$

We account for this by resetting the voq_cntr_{ij}s to the queue length(L_{queue}) when they get served. As shown in Figure 7, with this new weight, iHCF achieves 100% throughput under this traffic pattern in couple of iterations and behaves similar to OCF when run for higher number of iterations. The only added complexity from this variation is another set of counters that is required to count the queue length. The prediction mechanism of Section IV-B is still applicable since if the VOQ is served, the predicted value will be the queue length counter incremented by one. Since the voq_cntr_{ij} need to be reset to the queue length counter value, both of the counters need to be of the same size. Simulations show that counters of width $2log(N)$ bits are sufficient to provide good performance.

VI. Conclusion

The scheduler of an input-queued switch needs to provide good performance on a wide range of traffic patterns, yet remain simple for it to be implemented in a high-speed switch. In this paper, we presented a new algorithm iHCF, highest count first, which is a starvation-free maximal matching algorithm and performs well on all types of traffic patterns. Compared to other known algorithms which take into account the age of the cells as weights, iHCF provides a simpler implementation and better performance with only single iteration. iHCF incurs higher latency than iLPF, but still manages to achieve 100% throughput and the complexity is much simpler than iLPF. Compared to other well-known simple algorithms such as iSLIP, iHCF performs better on non-uniform traffic pattern with minimal additional complexity and yet maintains 100% throughput on uniform traffic.

By adding another set of counters, a variation to iHCF was also introduced which can handle traffic patterns that are not balanced across the inputs and the outputs. As part of future work, further analysis of the iHCF algorithm is needed to see if the variant proposed for iHCF in Section V can be theoretically proven to provide 100% throughput on all admissible traffic patterns. There are also other variants of the algorithms that can be investigated if different objectives of the scheduler is needed. For example, in the algorithm that we presented, all counters were assumed to be identical and behaved identically. However, by changing the individual counters such that they saturate at different values or by incrementing them by different values, preferential service can be given and QOS can be implemented.

References

[1] Demers, A., Keshav, S., Shenker, S. "Analysis and simulation of a fair queueing algorithm", *Journal of Internetworking Research and Experience*, pp 3-26, Oct. 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[2] Karol, M., Hluchyj, M., Morgan, S. "Input versus output queueing on a space division switch", *IEEE Trans. on Communications*, vol. 35, n. 12, Dec. 1987, pp. 1347-1356.

[3] Anderson, T.; Owicki, S.; Saxe, J.; and Thacker, C. "High speed switch scheduling for local area networks", ACM Trans. on Computer Systems. Nov 1993 pp. 319-352.

[4] McKeown, N. "Scheduling Algorithms for Input-Queued Cell Switches", PhD Thesis, University of California at Berkeley, May 1995.

[5] McKeown, N., Mekkittikul, A., Anantharam, V., Walrand, J. "Achieving 100% throughput in an input-queued switch", *IEEE Trans. on Communications*, vol. 47, n. 8, Aug. 1999, pp. 1260-1267.

[6] McKeown, N., Anderson, T. "A Quantitative Comparison of Scheduling Algorithms for Input-Queued Switches", Computer Networks and ISDN Systems, Vol 30, No 24, pp 2309-2326, December 1998.

[7] Mekkittikul, A., McKeown, N. "A Practical Scheduling Algorithm to Achieve 100in Input-Queued Switches", IEEE Infocom 98, Vol 2, pp. 792-799, April 1998, San Francisco.

[8] Mekkittikul, A. "Scheduling Non-uniform Traffic in High Speed Packet Switches and Routers", PhD Thesis, Stanford University, November 1998.

[9] Li, Y., Panwar, S., Chao J. "On the performance of a dual round-robin switch", IEEE Infocom'2001, vol. 3, April 2001, pp. 1688-1697.

[10] http://klamath.stanford.edu/tools/SIM/