An Evolutionary Approach to Optimal Web Proxy Cache Placement

Gwen Houtzager Christian Jacob Carey Williamson
Department of Computer Science
University of Calgary, Canada
$\{$*gwen,jacob,carey*$\}$*@cpsc.ucalgary.ca*

***Abstract*— This paper studies the Web proxy cache placement problem, in which m caching proxies are to be placed in a network so as to minimize the average response time for users accessing Web content. We compare an evolutionary approach to this network optimization problem with two classical approaches, namely dynamic programming and packet-level simulation. The results show that the evolutionary approach produces results as good as or better than the other approaches. Furthermore, the evolutionary approach is computationally faster, enabling the study of larger network scenarios than possible with the other approaches.**

I. Introduction

Given the explosion of Internet use in the last decade, much effort has been directed towards improving user-perceived performance on the World Wide Web.

One popular approach involves the installation of Web proxy caches, which provide a shared cache for a set of Web clients [19]. By exploiting commonalities in the Web browsing patterns of many users, Web proxy caching can reduce Internet traffic across a network, improve server responsiveness by reducing server load, and reduce user-perceived latency when accessing Web documents.

The strategic placement of Web proxies in a network can yield many performance advantages. Specifically, the goal is to keep request/response traffic off of slower inter-continental links that can inflate client response times. By reducing round trip delay, Web document downloads are faster for the user. Equally important, unnecessary network traffic is eliminated on busy Internet backbones.

The Web proxy cache placement problem is often formulated as an optimization problem: place m proxies within a network to minimize the mean user response time for retrieving Web objects. Solutions to this problem include graph theoretic, combinatorial, dynamic programming, and vector quantization approaches [6], [10], [11], [12], [13], [14].

One drawback of most theoretical approaches to the Web proxy cache placement problem is the limiting assumptions that are needed to make the problem tractable. For example, some approaches assume homogeneous clients, fixed-size documents, and identical hit ratios at each proxy cache. These assumptions differ from empirical observations of Web workload characteristics: Zipf-like distributions for client activity and Web object popularity [2], [5], heavy-tailed transfer size distributions [1], [5], and diminishing hit ratios at each successive level of cache due to filter effects [4], [16], [22]. Furthermore, theoretical approaches often ignore network protocol effects, such as bursty packet traffic, packet losses, and the dynamics of TCP flow control.

Another approach is to study the Web proxy cache placement problem from a network-layer perspective, at the packet level. For example, detailed packet-level simulations often use more realistic assumptions. Earlier simulation work [8], [18] has shown the impacts of network-level effects (e.g., round trip times, link speeds, network congestion, packet losses, TCP dynamics) on user-level Web performance.

The drawbacks of the packet-level simulation approach are the computational demands per simulation, and the number of simulations required. Conducting multiple simulations is possible for a small, simple Web proxy caching environment. However, applying the same “brute force” method to a moderately sized, complex network becomes infeasible.

In this paper we propose an evolutionary approach to the Web proxy cache placement problem. Through the evolutionary algorithm we can incorporate important assumptions about TCP flow control and Web caching effects, while maintaining the ability to study larger, more realistic network topologies. In essence, we provide a method for addressing the deficiencies of classical theoretical and simulation-based approaches to the problem.

We first compare results of the evolutionary approach to packet-level simulations and a dynamic programming algorithm for a small network model. We then apply the evolutionary computation techniques to a larger, more realistic network model, and discuss the results.

The rest of the paper is organized as follows. Section II provides some background on the cache placement problem, and briefly discusses prior work. Section III describes the experimental methodology for our work. Section IV briefly describes the classical approaches to this problem, while Section V introduces the evolutionary approach. Sections VI and VII present results for small and large network scenarios, respectively. Finally, Section VIII concludes the paper.

II. Background and Related Work

Web proxy caches provide a shared cache to a set of clients [19]. When a user requests a Web document from a particular origin server, the request goes via the proxy. If a valid copy of the document is cached at the proxy, then the user’s request is served from the proxy in the same way as it would have been served had it been handled by the origin server. Alternatively, if the document is not in the proxy cache, or if the document is not up to date, then the proxy forwards the request to the origin server. The origin server responds to the proxy, and the proxy forwards the response to the client. The proxy typically stores the document in its cache to serve future requests from other clients for the same document. To the origin server, the proxy appears and acts as a client making a request. To the client, the proxy appears and acts as the origin server responding to a request.

The placement of Web proxy caches (proxies) in a given network poses an interesting problem. Specifically, there is a theoretical, optimal solution to the placement problem where the number, size, and cost of Web caching appliances is minimized while the benefit of Web caching is maximized in terms of reducing user latency and bandwidth usage. In graph theory, this problem is referred to as the k-median problem, and has been proven to be NP-hard. As a result, current solutions to the problem rely on heuristic approaches and approximate models [10], [11], [12], [14].

Li *et al.* [13] have studied the problem of optimal placement of multiple Web proxies among potential sites, given a certain traffic pattern. The reduction in overall network traffic and the reduction in access latency are used to determine optimal placement. Using a dynamic programming approach, they propose an optimal solution for linear network topologies, and a heuristic approximation for tree topologies. Later work determined an optimal solution to the distributed caching problem in a network with a tree topology [14]. The latter paper provides the basis for the dynamic programming approach used in our work.

What makes the proxy placement problem more complicated than other instances of the k-median problem is the existence of upstream and downstream dependencies when evaluating potential proxy site locations. Empirical measurements show that the cache hit ratios tend to decrease at each successive level visited in a cache hierarchy [16]. This phenomenon is called the *cache filter effect* [22]. Packet-level simulations have shown that caching dependencies have a significant influence on the proxy placement problem [8].

The primary contribution in our paper is the application of an evolutionary algorithm to the Web proxy cache placement problem. To the best of our knowledge, this is a novel application domain for evolutionary computation.

III. Experimental Methodology

A. Network Model and Assumptions

The simple network model assumed for our study is shown in Figure 1. The network topology has a single Web server at the root (top) of an unbalanced tree, with six clients at the leaf level. The circular nodes, labeled P1 to P11, represent candidate proxy locations, at national, regional, and institutional levels. (Browser caches are ignored.) The lines represent routing paths from the server to the clients. Router nodes have buffers of size 100 packets with Drop Tail queueing. All network links have 10 Mbps transmission capacity. Link propagation delays are shown on the left edge of the diagram. The percentages beneath each client (square) indicate the proportion of the total Web request workload generated by each client. The numerical values adjacent to each link in Figure 1 represent weights used by a dynamic programming approach discussed in Section IV.

The network structure and the traffic workload in our model are intentionally unbalanced and asymmetric. The network represents a compromise: large enough so that the placement of proxies is not trivial, yet simple enough so that it can be easily studied and the results understood. Given this specific network topology, it is possible to determine an optimal placement for a set of m Web proxies using dynamic programming, packet-level simulation, and an evolutionary approach. The simple model facilitates the analysis and comparison of results from these three methods.

The larger network model used in our study follows the same assumptions outlined for the simple network model. The larger network is discussed in more detail in Section VII.

B. Simple Network Web Workload Model

The Web workload in our study is synthetically generated using WebTraff [17], a Web workload modeling tool. The experiments use a workload of 100,000 requests. Each line in the workload file represents the download of a Web object by one of the clients. The workload file format has four columns: a timestamp (request arrival time); a source node (provider of the Web object); a sink node (the requesting client); and a transfer size (in bytes). The request arrival process is Poisson [1], with a specified mean arrival rate (e.g., 30 requests per second). The transfer size is drawn from a hybrid distribution, with a log-normal body, and a Pareto tail. The median transfer size is 1 KB (2 TCP packets), while the mean is 8 KB (16 packets). The largest transfer is 3 MB. The source and sink for each Web transfer are chosen at random according to the client request rates and the cache hit ratios being modeled. The default source node for each Web transfer is the origin server.

C. Experimental Factors and Performance Metrics

The primary experimental factors in our study are the number of proxies to be placed in the network, and the hit ratios for the Web proxy caches.

1) Number of Proxies: In the simple network scenario, there are $n = 11$ candidate locations for Web proxies. There are two obvious performance bounds for this network. The *worst* possible case occurs when no proxies are placed in the network. The *best* possible case occurs when 11 proxies are used. In our experiments, the number of proxies to be placed in the network is set from 1 to 11. In the larger network model, the number of candidate proxy locations is 160, therefore the number of proxies to be placed can be set from 1 to 160. This network is discussed in Section VII.

2) Cache Hit Ratio: The cache hit ratio is defined as the proportion of client requests that a given proxy is able to satisfy from its cache. The hit ratio is expressed as a percentage of the total client requests handled by that proxy.

Table I shows the cache hit ratio values used for both the simple and complex network models in our study. These ratios were designed to represent typical Web proxy cache

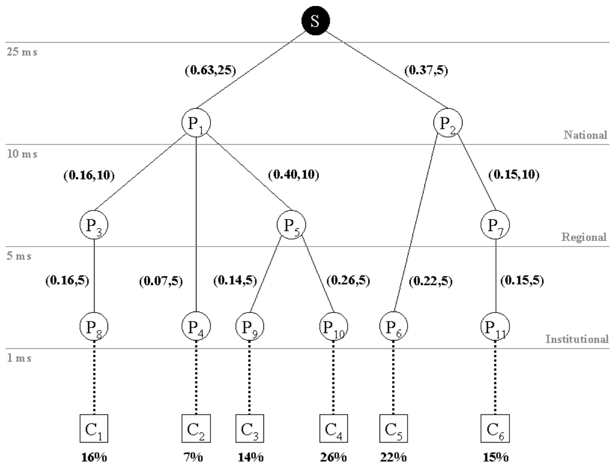

Fig. 1. Network topology and workload characteristics

hit rates, as well as the "diminishing returns" cache filter effect reported in the literature [16], [21], [22].

TABLE I

PROXY CACHE HIT RATIOS ASSUMED FOR EXPERIMENTS

# Proxies on Client-Server Path	Cache Hit Ratios		
	Level 1 Institutional	Level 2 Regional	Level 3 National
1	30%	–	–
1	–	20%	–
1	–	–	15%
2	30%	15%	–
2	30%	–	10%
2	–	20%	10%
3	30%	15%	7.5%

The cache hit ratios diminish as the network caching level increases from the institutional level to the national level. Since higher level proxies serve more diverse client groups using a finite shared cache, the probability of an individual client finding a requested document at a higher level proxy decreases. Cache hit ratios were assigned consistently across caching levels for each client stream.

The primary performance metric of interest is the mean user-perceived transfer time for Web object downloads.

IV. Traditional Approaches

A. Dynamic Programming

The dynamic programming (DP) approach used in our study is based on work by Li *et al.* [13], [14]. The approach assumes that the network is a directed graph of nodes and edges. Each node has an associated weight that represents the volume of traffic in the absence of proxies. Within the network tree, a set of nodes are chosen as proxy sites. Each proxy set solution has an associated cost, and the set that minimizes the cost represents the optimal solution.

The dynamic programming algorithm is computationally efficient. The static costs are pre-computed for small instances of the proxy placement problem, and then stored in a global array, indexed by the number of proxies and the subtree in which the proxies are placed. Solutions for larger instances of the problem are then built in a table-driven fashion using these pre-computed values. The running time of the algorithm is $O(n^3m^3)$. Further details on the DP approach are provided in [7], [8], [13], [14].

B. The Packet-Level Simulation Approach

The packet-level simulation technique in this paper follows the same approach as prior work [8]. It represents a "brute force" exhaustive search of all possible proxy set combinations. Given n candidate proxy locations, and $m \leq n$ proxies to be placed, the number of possible combinations is:

$$C(n,m) = \frac{n!}{m!\;(n-m)!}$$

For example, for the 11-node network in Figure 1, there are $C(11,3)$ = 165 possible ways to place 3 proxies. We use the *ns-2* network simulator [3] to carry out the simulations.

V. The Evolutionary Approach

Figure 2 provides a structural overview of the evolutionary algorithm used. The evolutionary approach uses parameters consistent with those of the simulation experiments. Initially, candidate proxy locations are randomly selected. Through the process of mutation and recombination, the initial proxy set is grown into a population of proxy sets, each one evaluated for its "fitness for purpose". The fittest member of a population advances to the next generation, where the process is repeated. The algorithm terminates when the specified convergence criterion is reached [9].

A. Initialization

At each iteration of the evolutionary algorithm, a group of 'parent' proxy set combinations 'breed' to produce a population that includes both the initial parents and their offspring (children). To initialize the parent population for generation 0, a randomly selected set of m proxy sites is assigned. More specifically, an array is populated with a randomly chosen set of integer values. The length of the array represents the number of proxies to be placed in the

```
program EA {
    t = 0;
    initialize population P[t];
    until done do {
        t = t + 1;
        parent_selection P[t];
        recombine P[t];
        mutate P[t];
        evaluate P[t];
        survive P[t];
     }
 }
```

Fig. 2. Structural overview of the evolutionary algorithm

network, while each integer value within the array represents the arbitrary numerical label associated with a candidate proxy site. Next, the parent arrays undergo recombination and mutation (to be explained next), in order to produce child arrays with similar structure. The number of parents and children is specified by the program parameters. Combined, the parents and children represent the population in a given generation. Each time the evolutionary algorithm is run, a different initial set of random parents is generated.

B. Genetic Operators

The evolutionary algorithm uses two genetic operators called *recombination* and *mutation*. These two genetic operators serve different purposes. Recombination combines existing knowledge from different individuals already in the population, while mutation randomly introduces new, potentially better values (proxies) into the population.

1) Recombination: Inspired by biological reproduction, recombination refers to the re-ordering and sharing of existing genetic information from the parents. More specifically, the child arrays are produced by swapping the parent information according to a chosen crossover point (one-point crossover). The result of recombining two parent arrays is two new child arrays. Each child array contains a portion of the proxy sites from one parent array and the remaining proxy sites from the other parent array. Therefore, the child array represents a new proxy set combination. There is a globally specified probability of recombination occurring for each individual. When recombination occurs, a crossover point is determined uniformly at random. Figure 3 provides an overview of the recombination operator.

```
crossOver( parent1, parent2 ) {
    prob1 = random();
    if( prob1 < crossRate )
    do {
        xPoint = random();
        for i = 1 to xPoint-1
           child1[i] = parent1[i];
           child2[i] = parent2[i];
        for j = xPoint to numProxy
           child1[j] = parent2[j];
           child2[j] = parent1[j];
     }
 }
```

Fig. 3. The recombination operator algorithm

Given the simplistic nature of the recombination operator, a child could inherit the *same* proxy location from *both* parents, resulting in a duplicate entry in the child array. The evolutionary algorithm does not preclude this possibility, nor does it need to. Intuitively, a proxy set combination with duplicate entries will have inferior performance, since it leaves some other candidate location without a proxy. If the duplication is not eliminated by mutation or recombination, then the child is unlikely to survive from one generation to the next. As a result, the algorithm can ignore the issue of duplication.

2) Mutation: In nature, mutation refers to the process of randomly changing the genetic makeup of an organism. Within the context of the proxy placement problem, mutation refers to the process of changing the proxy set combination represented in an array.

During the evolution phase of the algorithm, a mutation operator is applied to each array within a population. First, the mutation rate determines whether a single proxy site within a proxy set will change. Next, the step size determines the mutability of the item. That is, the step size, which is drawn from a Gaussian distribution, determines the numerical *amount* by which a selected array item can change. Figure 4 provides an overview of the mutation operator.

```
mutation( child ) {
    for i = 1 to numProxy do {
        prob1 = random();
        if( prob1 < mutationRate ) do {
           step = stepSize * Gaussian();
           child[i] = child[i] + step;
         }
     }
 }
```

Fig. 4. The mutation operator algorithm

C. Evaluation Function

The evaluation function, also known as the fitness test, is an important component of an evolutionary algorithm. This function distinguishes between different proxy set combinations, and determines the fittest in a given population. Our evaluation function uses the same parameters as the packet-level simulation study. This consistency facilitates comparison of results from different solution approaches.

For the proxy placement problem, the key is to minimize the transfer time in order to reduce the user-perceived latency. For this purpose, the expected overall network transmission time $E(T)$ is a suitable fitness metric. The transfer time is the sum of the transmission time that it takes to transfer packets across the network plus any queueing delays plus any additional transmissions caused by packet loss. In this study, queueing delays and packet loss are ignored. However, their inclusion requires only a slight modification to the fitness equation.

For a three-level caching hierarchy, the average transfer time depends on the hit ratio HR for each level of cache, and the average transfer time from each cache:

$$E(T) = HR_1(T(P_{institutional})) + HR_2(T(P_{regional})) + HR_3(T(P_{national})) + (1 - \sum HR)(T(P_{server}))$$

Furthermore, the transfer time for an individual client request with N packets can be approximated by the following formula [15]. The formula incorporates the effects of TCP slow start, as well as the network round trip time RTT.

$$T(i) = (1 + \lfloor log_2 N \rfloor) * RTTi$$

The overall fitness value for the evolutionary algorithm depends on the expected transfer times for clients, weighted by the proportion r of traffic generated by each client:

$$FitnessValue = \sum_{i=1}^{c} r_i(E(T_i))$$

The numerical result of the fitness test is used in relative terms to rank and compare proxy set combinations within the population.

D. Parent Selection

Survival from one generation to the next is implemented using one of two deterministic methods, following the selection schemes described for *Evolution Strategies* [20]. In the *Comma* strategy, after evaluating a population, the K best children survive and replace the parents of the current generation. Parents do not survive from one generation to the next. In the *Plus* strategy, the K best individuals (child and parent alike) survive to the next generation. Both strategies are considered in this analysis.

VI. Results for Small Network Model

Table II provides a comparison between the evolutionary results and the results from the packet-level simulations and dynamic programming algorithm. For example, when placing a single proxy, both the evolutionary algorithm and the packet-level simulation recommend proxy site P10, while the dynamic programming approach recommends site P1. The average transfer time and rank (as determined from the simulation experiment) for each solution are shown in the right-hand portion of the table.

The results from the evolutionary algorithm are almost identical to those from the simulation experiments. The only difference occurs for $m = 3$ proxies where the evolutionary algorithm chose P5 as the third proxy site and the simulation results indicate that a proxy at P1 is the optimal location (Figure 5). Note that the set (P5, P6, P10) was the second-best configuration (rank 2) from the simulations, and the performance difference between the two choices is only 1 millisecond. Although not shown in the table, beyond $m = 4$, the optimal proxy sets in both evolutionary and simulation results are the same [7].

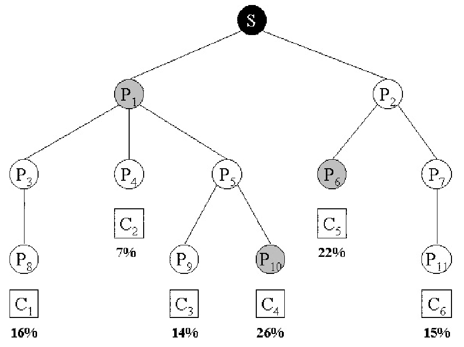

Fig. 5. Optimal Proxy Set for $m = 3$

The evolutionary results improve noticeably upon the results from the dynamic programming approach. The major reason for this is that the dynamic programming approach ignores TCP dynamics (such as slow start), cache hit ratios, and cache filtering effects [8]. These effects are all considered in the evolutionary algorithm.

Computationally, the simple network model is a trivial problem for the evolutionary algorithm. The algorithm quickly converges to an optimal solution in a few seconds, requiring only 10 to 20 generations of evolution. Packet-level simulations for the same network required extensive time and computational effort (i.e., multiple simulations, each 5-7 minutes in duration).

The experimental results from the evolutionary approach on the simple network model are encouraging. However, a larger, more realistic network model is required to better test the capabilities of the evolutionary algorithm.

VII. Results for Large Network Model

Figure 7 shows the larger network model used in this study. It consists of a single server, 130 clients, and 160 candidate proxy locations. The network uses three hierarchical caching layers, with 113 institutional caches, 36 regional caches, and 11 national caches. The structure of the network was randomly generated. The network is an unbalanced tree with non-uniform traffic patterns. Each proxy node has between 2 and 7 clients. To spread the overall Web request load among the clients in the network, random request counts were assigned to each client. The maximum count was 106, the minimum 1, the mean was 80, and the median 83.

A. Discussion of Results

Figure 6 provides an overview of the results for the evolutionary algorithm. The graph shows the fitness function value as it behaves over 150 generations. The fitness function value is averaged over 10 separate trial runs in each case. As expected, it improves (decreases) over time. Each line on

TABLE II

COMPARISON OF RESULTS FOR DYNAMIC PROGRAMMING (DP), SIMULATION (SIM), AND EVOLUTIONARY ALGORITHM (EA)

Num Proxies m	Optimal Proxy Set Solution			Relative Rank			Mean Transfer Time (sec)		
	DP	SIM	EA	DP	SIM	EA	DP	SIM	EA
1	{1}	{10}	{10}	3	1	1	0.256	0.252	0.252
2	{1,2}	{6,10}	{6,10}	17	1	1	0.245	0.236	0.236
3	{1,2,5}	{1,6,10}	{5,6,10}	31	1	2	0.230	0.223	0.222
4	{1,2,5,6}	{5,6,8,10}	{5,6,8,10}	48	1	1	0.218	0.210	0.210

the graph represents results for a different value for m (the number of proxies placed in the network).

The top line in the graph represents the case when only 5% of the nodes in the overall network are designated as proxy sites (9 in total). The solution set for $m = 9$ proxies determined by the evolutionary algorithm consisted of 6 national level caches and 3 regional caches. The regional caches were those along the routes to the busiest clients. Since this scenario contains the fewest proxies, the overall mean transfer time (and hence the fitness value) was the highest (worst) compared with the others. The evolutionary algorithm converges towards a solution quickly and consistently, as evidenced by the stable fitness values after 50 generations.

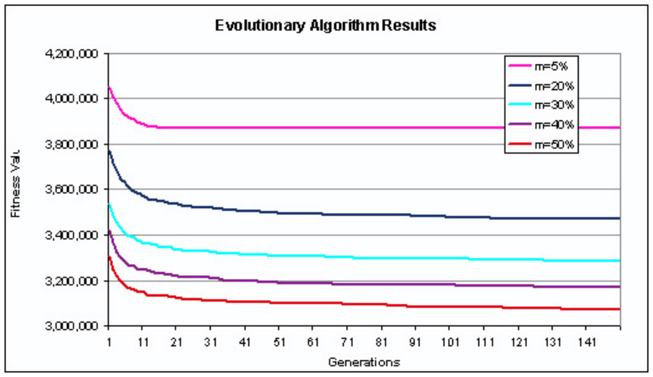

Fig. 6. Overview of evolutionary algorithm results

The bottom line in the graph represents the case when 50% of the network nodes are selected as proxy sites (80 proxies). The decline for this curve is more gradual than in the $m = 5\%$ case. The fitness function continues to decrease in value even after 200 generations, albeit at a much slower rate than initially.

Figure 7 shows an overview of the network and the proxy placement configuration determined by the evolutionary algorithm. The solution set contains 45 institutional caches, 24 regional caches, and all 11 national level caches.

While we cannot verify the optimality of this configuration, we can make several qualitative observations about its structure. Without exception, the sites chosen for institutional caches are all near the busiest clients. For example, the cluster circled in the upper left corner of Figure 7 represents the largest subtree and contains a relatively large proportion of the busiest clients. As expected, this cluster contains a higher proportion of proxies. Similarly, the cluster highlighted by the dashed box on the right represents the subtree containing the fewest busy clients. In this case, the cluster contains few proxies. The regional caches tend to be placed where the subtree contains many nodes or where there are fewer proxies located at the level below. Interestingly, for the regional and institutional levels, no proxies are placed along routes leading to low-traffic clients (clusters highlighted by the smaller dashed circles in Figure 7).

The foregoing solution was determined by running the evolutionary algorithm for 900 generations (half hour run time). Each generation included a population of 500 children and 50 parents. Initially, the mutation rate was set at 30% with a step size of 40. The probability of recombination was 50%. Both genetic operations were made highly probable so as to consider a broad set of potential proxy locations. After the evolutionary algorithm had run for 300 generations, the genetic operator probabilities were modified. The mutation rate was reduced to 5% with a step size of 5, and the probability of recombination was also reduced to 5%. At this point, the important proxy locations had already been identified by the algorithm. Therefore, to encourage convergence towards the optimal solution, consideration of radically different proxy set combinations was no longer necessary. Finally, after 600 generations, the genetic operator values were reduced to very low levels. The mutation rate was set to 1% with a step size of 2, and the probability of recombination was set to 1%. At this point, only single elements within the array data structure would be tweaked, if at all. Therefore, assuming that the solution thus far was generally correct, a single poor choice for a proxy location could be corrected if necessary. A detailed sensitivity analysis of program parameters follows in the next section.

B. Sensitivity of Program Parameters

In evolutionary computation, there is never a single "correct" setting for the genetic operators and program parameters. To broaden the scope of the evolutionary experiments, we varied evolutionary control parameters in an attempt to understand sensitivities in the results. In particular, we consider the effects of the mutation rate, step size, selection strategy, and recombination rate.

1) Mutation Rate: Figure 8 shows the behaviour of the evolutionary algorithm under different mutation rates (all other program parameters remaining constant). The fitness

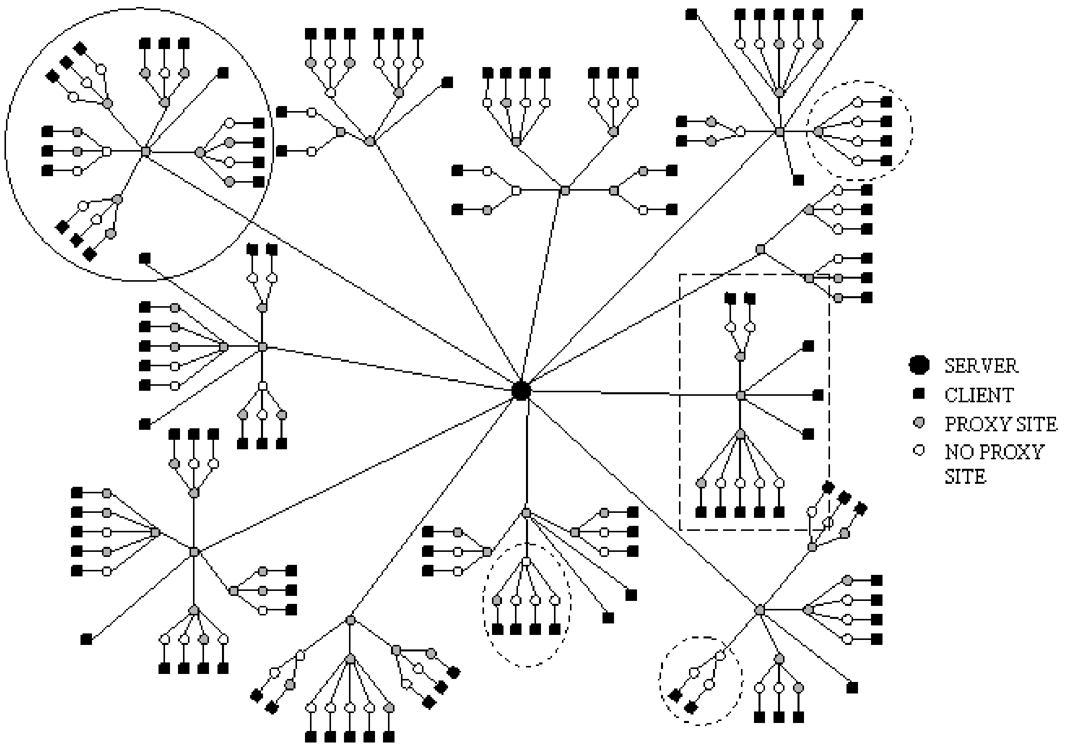

Fig. 7. Network topology and evolutionary algorithm results for $m = 80$ proxies in large network model

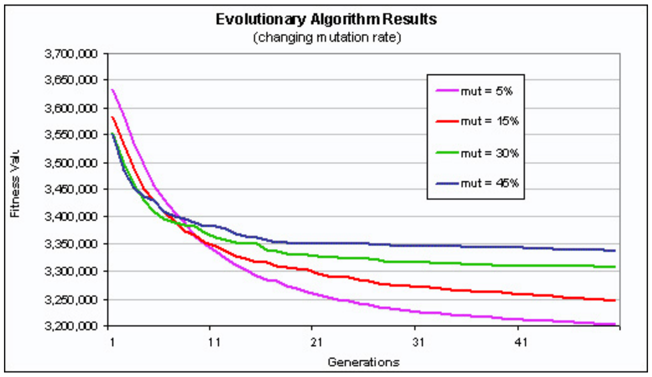

Fig. 8. Evolutionary algorithm sensitivity to mutation rate

function value for each mutation rate represents the average of 10 trial runs. Initially, higher mutation rates appear to speed up the process of finding better solutions. However, at a certain point, as the algorithm starts to converge towards a good solution, a higher mutation rate perturbs the algorithm, preventing the population from 'landing' on the optimal set of proxies. The mutation rate of 30% seems to perform best initially. However, in the long run, beyond 10 generations, the evolutionary algorithm behaves best with a lower mutation rate of 5%.

2) Mutation Step Size: Figure 9 shows the behaviour of the evolutionary algorithm for different mutation step sizes. The step size represents the mean of a Gaussian distribution. Similar to the mutation rate operator, a somewhat larger step size (e.g., 40 or 60) produces faster initial improvements, but the earlier gains disappear as the generations continue. Oddly, the higher step sizes (e.g., 60 and 90) do not seem to disrupt the evolutionary algorithm in the long run, which is an unusual finding. Typically, low mutability is associated with fine tuning to a local or global optimum. One explanation for the absence of this phenomenon is that in our model, the numerical proxy labels do not reflect topological location. That is, a nearby proxy may have a vastly different *numerical* label and hence would require a larger step size to be found randomly by the evolutionary algorithm.

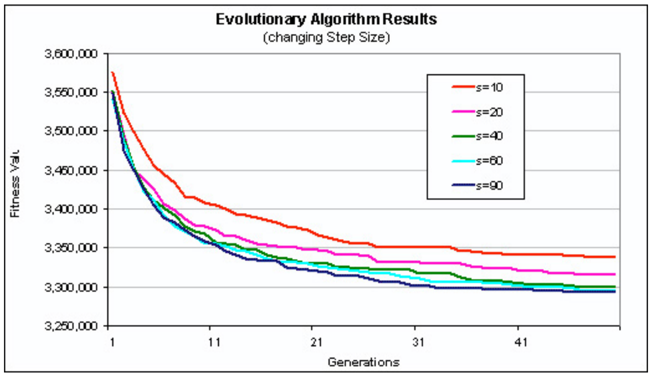

Fig. 9. Evolutionary algorithm sensitivity to step size

3) Selection Strategy: When the parents of a population are not included as part of the next generation, the resulting children may not be as *fit*. As a result, the evolutionary process will go 'backwards' for a short while, so to speak. In nature, parents normally die before their offspring, and there

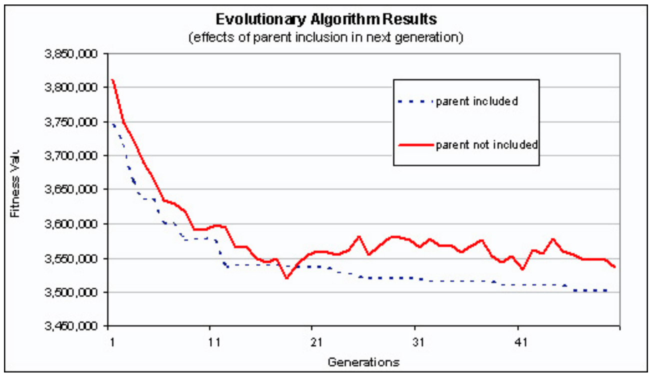

Fig. 10. Evolutionary algorithm sensitivity to parent selection

can be deterioration from one generation to another. In fact, this is an important means of preventing a group or species from ‘freezing’ genetically. However, excluding parents from a population in successive generations can sometimes prevent an evolutionary algorithm from converging.

Figure 10 shows the fitness function value over 50 generations for the case in which parent solution sets are excluded from the surviving group, and for the case in which parent sets are included in the selection process. The line representing the ‘parent exclusion’ case is more jagged. If the parent population was included among the survivors, the fitness value behaves more consistently.

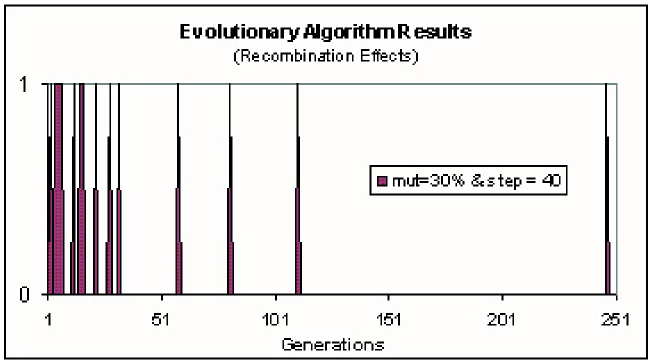

Fig. 11. Prevalence of the recombination operation over 256 generations

4) Recombination: Figure 11 presents a time series plot that shows the influence of the recombination operator. Recall that the recombination operator is applied with a certain probability, which means that there is an on/off impulse property to its effect on the behaviour of the evolutionary algorithm and on the population. The graph shows the generations in which the fittest individual emerged as a result of the recombination of parental sets. Initially, the recombination operator is dominant, as shown early in the graph. However, as the best solution set in a given generation approaches the optimal solution over time, recombination is rare, and the mutation operator becomes the most influential.

VIII. Summary and Conclusions

This paper proposed an evolutionary approach to the Web proxy placement problem. The approach was applied to a simple network and the results compared to packet-level simulation experiments and a dynamic programming algorithm. The results look very promising.

The dynamic programming algorithm has the advantage of being scalable to larger networks. However, the optimal proxy sets from the dynamic programming algorithm did not match those determined by simulation experiments. The most likely explanation is that the dynamic programming method considers only network traffic volume and link transmission latency. It does not consider network influences such as cache filtering effects and TCP dynamics. The packet-level simulation experiments show that these network influences are significant.

While the combinatorial simulation approach to the proxy placement problem guarantees an optimal solution for a given network topology and client workload (since all possible cases are simulated), the effort required becomes prohibitive as the size of the network and the number of proxies grow in scale. Unfortunately, for most practical network topologies, this approach is not economically or computationally viable.

The evolutionary approach provides the best of both worlds. The results for the simple network closely match those determined by the packet-level simulations, indicating that the fitness function captures the essential elements of the network and caching implications. As well, the evolutionary approach can handle much larger network topologies, although for these networks the proxy placement solution cannot be proven optimal. Fortunately, given the many uncertainties about Web workloads and TCP dynamics in a large internetwork, robust *good* solutions are more desirable than perfectly *optimal* solutions [8]. The evolutionary approach can quickly find good solutions to the proxy placement problem, dramatically narrowing the search space for subsequent optimization efforts.

Acknowledgements

Financial support for this research was provided by iCORE (Informatics Circle of Research Excellence) in the Province of Alberta, and by the Natural Sciences and Engineering Research Council of Canada, through NSERC Research Grant OGP0121969 and an NSERC Postgraduate Scholarship.

References

[1] M. Arlitt and C. Williamson, “Internet Web Servers: Workload Characterization and Performance Implications”, *IEEE/ACM Trans. Networking*, Vol. 5, No. 5, pp. 631-645, October 1997.
[2] L. Breslau, P. Cao, G. Phillips, and S. Shenker, “Web Caching and Zipf-like Distributions: Evidence and Implications”, *Proceedings of IEEE INFOCOM*, pp. 126-134, March 1999.
[3] L. Breslau *et al.*, “Advances in Network Simulation”, *IEEE Computer*, Vol. 28, No. 5, pp. 59-67, May 2000.
[4] M. Busari and C. Williamson, “Simulation Evaluation of a Heterogeneous Web Proxy Caching Hierarchy”, *Proceedings of IEEE MASCOTS*, pp. 379-388, 2001.
[5] M. Busari and C. Williamson, “On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics”, *Proceedings of IEEE INFOCOM*, Vol. 3 pp. 1225-1234, 2001.

[6] C. Cameron, S. Low, and D. Wei, "High Density Model for Server Allocation and Placement", *Proceedings of ACM SIGMETRICS*, pp. 152-159, June 2002.

[7] G. Houtzager, *Optimizing Web Proxy Cache Placement and Performance*, M.Sc. Thesis, University of Calgary, 2005.

[8] G. Houtzager and C. Williamson, "A Packet-Level Simulation Study of Optimal Web Proxy Cache Placement", *Proceedings of IEEE MASCOTS*, pp. 324-333, October 2003.

[9] C. Jacob, *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann Publishers, 2001.

[10] X. Jia, D. Li, X. Hu, H. Huang, and D. Du, "Optimal Placement of Proxies of Replicated Web Servers in the Internet", *Proceedings of* 1^{st} *Int'l Conference on Web Information Systems Engineering (WISE)*, Vol. 1, pp. 55-59, 2000.

[11] X. Jia, D. Li, and X. Hu, "Placement of Read-Write Web Proxies in the Internet", *Proceedings of IEEE ICDCS*, pp. 687-690, April 2001.

[12] P. Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem", *ACM Transactions on Networking*, Vol. 8, No. 5, pp. 568-582, October 2000.

[13] B. Li, M. Golin, X. Deng, and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet: Linear Topology", *8th IFIP Conference on High Performance Networking*, September 1998.

[14] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet", *Proceedings of IEEE INFOCOM*, Vol. 3, pp. 1282-1290, March 1999.

[15] Y. Li and C. Williamson, "A Hysteresis Model for Web/TCP Transfer Latency", *Proceedings of IEEE MASCOTS*, pp. 167-174, October 2004.

[16] A. Mahanti, C. Williamson, and D. Eager, "Traffic Analysis of a Web Proxy Caching Hierarchy", *IEEE Network*, Vol. 4, No. 3 pp. 16-23, May/June 2000.

[17] N. Markatchev and C. Williamson, "WebTraff: A GUI for Web Proxy Cache Workload Modeling and Analysis", *Proceedings of IEEE MASCOTS*, pp. 356-363, October 2002.

[18] N. Markatchev and C. Williamson, "Network-Level Impacts on User-Level Web Performance", *Proceedings of SCS SPECTS*, July 2003.

[19] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Addison-Wesley Press, 2001.

[20] I. Rechenberg, *Evolutionsstrategie:Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[21] P. Rodriguez, C. Spanner, and E. Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching", *Proceedings of 4th International Web Caching Workshop*, pp. 37-48, April 1999.

[22] C. Williamson, "On Filter Effects in Web Caching Hierarchies", *ACM Transactions on Internet Technology*, Vol. 2, No. 1, pp. 47-77, February 2002.