OPTIMIZATION ALGORITHMS FOR PROXY PLACEMENT IN CONTENT DISTRIBUTION NETWORKS

by

JUN WU

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2011

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Kaliappa Ravindran, Ph.D.

_____ _____

Date Chair of Examining Committee

Theodore Brown, Ph.D.

_____ _____

Date Executive Officer

Octavio Betancourt, Ph.D.

Akira Kawaguchi, Ph.D.

Dajin Wang, Ph.D.

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

OPTIMIZATION ALGORITHMS FOR PROXY PLACEMENT IN CONTENT DISTRIBUTION NETWORK

by

JUN WU

Adviser: Professor Kaliappa Ravindran

Popular web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. One of the solutions is to using content distribution network (CDN). When we optimize proxy placement in content distribution network we want to maximize the user experience in the mean time to minimize the resource consumption. We are dealing with a multi-objective optimization problem for CDNs in the thesis: the content latency experienced by users and the system resources expended to deliver content. After the topology model for content distribution network, we listed metrics to evaluate content distribution network. Then we defined the objective function for optimizing proxy placement. After reviewing the existing proxy placement algorithms we used genetic algorithm to solve the proxy placement problem. We first

apply the genetic algorithms to a simple network, and find that the results are better than greedy algorithm and identical to the global optimal. Then we apply the genetic algorithm to a realistically larger network topology. The genetic algorithm can find good solutions to the proxy placement problem. But compared to greedy algorithm, it is less efficient. So when response time is important, genetic algorithm is not the best choice. To optimize the genetic algorithm we found if we choose higher mutation rate and step size at the beginning of the genetic operation and decrease the mutation rate and step size as we go through the genetic operation, the results are much better. For the initial populations, if we want quick response to the sudden change of client access we can choose smaller initial population, if we have sufficient response time especially in initial setting up the network we should choose larger initial populations. Under optimized genetic algorithm, we experiment three different network examples with their sizes between the simple network and large network. Since the network sizes are smaller we are able to run the exhaustive search to find the global optimal. In all cases the results from genetic algorithm outperform greed algorithm and close to global optimal. Although it is not guaranteed to find the global optimal solution as the simple network did, a robust good solution is enough to solve real world problems.

Acknowledgements

I would like to express my deepest thank and appreciation to my advisor Prof. Kaliappa Ravindran. His encouragement and support made this possible.

I wish to thank Prof. Octavio Betancourt and Prof. Akira Kawaguchi for a number of fruitful discussions and stimulating conversations.

Special thank to Prof. Dajin Wang for accepting being on the supervisory committee despite his busy schedule.

Most of all I owe a tremendous debt of gratitude to my parents Fengxuan Xie and Tianyao Wu. Without their unconditional love, it is not possible to go through the difficult time of my studying in the Ph.D. program.

Table of Contents

List of Tables

List of Figures

1. Introduction

With the explosive growth of the World Wide Web, popular web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. Content distribution network (CDN) has been used for this purpose; it is a group of geographically dispersed servers deployed to facilitate the distribution of information generated by Web publishers in a timely and efficient manner.

CDN has two advantages. The first advantage is that it can reduce Client's latency and network traffic by redirecting client requests to a proxy closest to that client. The second advantage is that it can also improve the availability of the system, as the failure of one proxy does not result in entire service outage. Both advantages result from the fact that making multiple copies of the content of a web publisher and distributing that content across the Internet removes the necessity for customer requests traversing a large number of routers to directly access the facilities of the web publisher. This reduces traffic routed through the Internet as well as the delays associated with the routing of traffic.

When we optimize proxy placement in content distribution network we want to maximize the user experience in the mean time to minimize the resource consumption. These two objects are conflict with each other. To

maximize the user experience we need to place proxy as many as possible, but that will increase the resource consumption. To minimize the resource consumption we need to minimize the number of proxy placement and that will increase the user latency so the user experience will decrease. We are dealing with a multi-objective optimization problem here. The research community has seldom addressed proxy placement and only few solutions have been proposed. We propose to use genetic algorithm to solve this multi-objective optimization problem.

The organization of the dissertation is as follows. Chapter 2 we propose our topology model for content distribution network. After listing metrics to evaluate content distribution network in chapter 3, we give objective function in optimization in proxy placement in chapter 4. In chapter 5 we listed existing proxy placement algorithms. Chapter 6 is about using genetic algorithm to solve the proxy placement problem. Chapter 7, 8 give Simulation result for a simple topology and simulation results for a large realistic topology respectively. Chapter 9 discusses how to optimize the genetic algorithm parameters. Chapter 10 gives simulation results for more topologies. Finally chapter 11 concludes the dissertation.

2. Topology model for content distribution network

The infrastructure-level topology of a CDN is modeled as a graph $G(V,E)$, where V is the set of in-network processing nodes and E is the set of edges interconnecting these nodes by network links. The clients reside in a set of nodes Vc, and access the content hosted on a remote master server node M. All the remaining nodes in the topology ($V - M - Vc$) are deemed as proxy-capable. A proxy placement algorithm chooses one or more of these nodes to host the proxies of M in a way to service the clients in a cost-optimal manner. See Figure 1 for a layered view of the proxy placement functions in a CDN.

2.1 Distribution tree overlays

The nodes hosting the proxies, Vr, are part of a content distribution tree $T(\{M, Vc, V'\},E')$ which has the root node at M connected to the leaf nodes at Vc through a set of nodes V' and edges E'. The tree T is basically an overlay set up on the infrastructure topology $G(V,E)$ with a set of nodes $\{M, Vc, V'\}$ and edges E',

Where $E' \subseteq E$, $V' \subseteq (V-\{M, Vc\})$ and $Vr \subseteq V'$. Refer to Figure 1.

Given a server-client configuration *{M, Vc}* placed in the infrastructure topology *G(V,E)*, we assume that the CDN service provider (SP) first runs an overlay multicast routing protocol, such as the variants of DVMRP(RFC 1075) and MOSPF (RFC 1584), to set up the tree *T({M, Vc, V'},E')*. At the routing level,

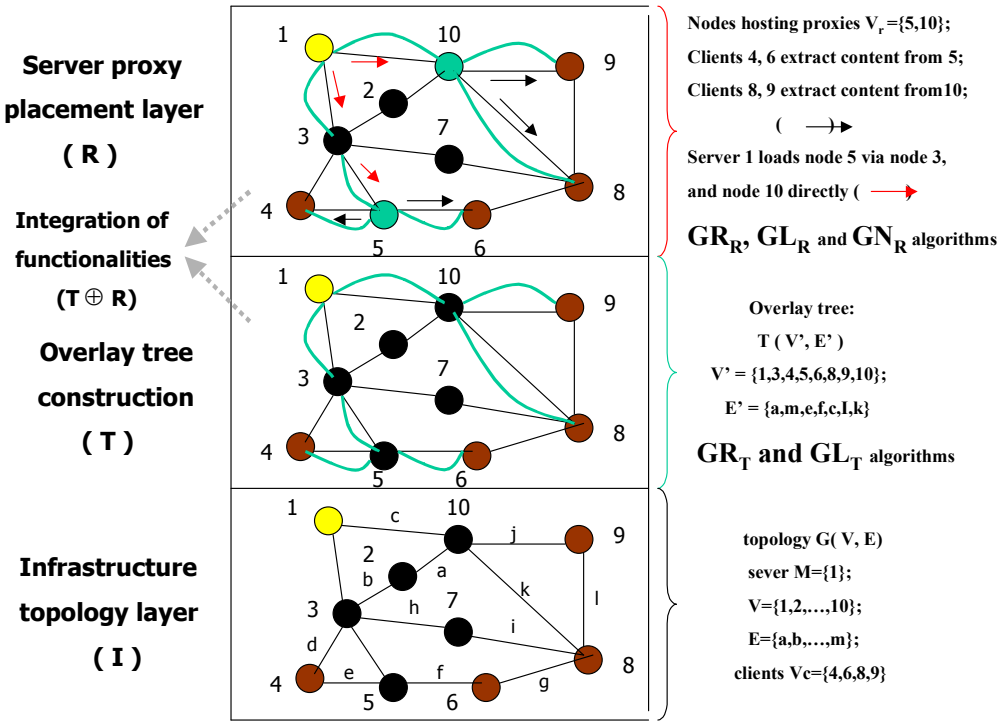


Figure 1. Layered view of replica placement functions in a CDN

the tree *T* may be globally optimal or incrementally optimal based on the tree setup algorithm chosen, where the optimality is measured in terms of the total number of hops in the interconnection paths between *Vc* and *M*.

After the overlay tree *T* is set up (regardless of how optimal *T* is at the routing level), the SP runs a proxy placement algorithm to select the nodes $V_r \subseteq V'$ to host the proxies in a cost-optimal manner. With the proxies *Vr* placed on the tree *T*, each client has exactly one path to reach its content serving node: which may be a proxy for *M* or the master server *M* itself. Here, the proxy placement determines the path length faced by various clients in accessing content, and hence impacts the QoS experienced in the CDN service interface to the clientele, namely, the content access latency experienced by individual clients.

2.2 Engineering considerations for proxy placement

The optimality of proxy placement is prescribed in terms of a cost metric associated with content distribution to the clients by the SP. One possible metric is the total bandwidth consumed for content delivery across all clients. Another metric may be the QoS degradation experienced by individual clients due to excessive content access latency (which translates to a revenue loss suffered by the SP). All such cost metrics are computable from the length of path segments from various clients to their content serving nodes and/or the path characteristics (such as link delays & capacity, link congestion, etc).

Furthermore, each additional proxy placed on the tree incurs a system-wide cost. Unlike the QoS-oriented costs, the system cost may involve loading the content in the proxy nodes, installing software agents in these nodes to service the clients, and allocating the node-internal resources (such as disk storage, CPUs, and memory). The system cost, in conjunction with the QoS needs of clients, determines the number of proxies $|Vr|$ and their placement.

For read-only content (e.g., the President's speech, public health statistics, and newspaper columnist reports), the number of proxies and their placement, as given by Vr, are determined by the cost considerations arising from the user-level QoS and the system-level performance. For contents that can change (e.g., online news updates), Vr is also determined by the overheads incurred to keep the proxies up-to-date with respect to the master server M, namely, messages exchanged for proxy updates and synchronization. In our research, we consider read-only contents.

2.3 Elements of cost models

Since the dissertation purports to expose the algorithmic issues in proxy placement, examining the issues for read-only content keeps our cost models and algorithm formulations simpler while not compromising the

generality of the models. Typical metrics relevant at the user and system levels are:

User-level: Content access latency and content availability;

The access latency experienced by a client *C* depends on the packet delays and the available bandwidth in the path connecting *C* to its nearest proxy node *P*. Larger latencies reduce the QoS seen by *C*, and hence lower the SP's revenues. If the access latency exceeds a user-specified limit Δ, the content is deemed as unavailable to *C*. Even in the absence of hard failures (such as node crashes and link outages), the latency can become high due to link congestions and node overloads--which may arise due to the sharing of a path segment by *C* with many other clients. The content availability measure, which is based on how often the access latency at *C* exceeds Δ, is thus tied to the resource allocation and sharing in the path segment leading to *P* (assuming that there are no hard failures in the path).

System-level: Bandwidth usage in the path;

The amount of data transported from the proxy node *P* to

deliver content to the client C determines the bandwidth usage. Even if other clients share this path segment with C, the asynchronous nature of content demands by various clients increases the bandwidth usage in proportion to the number of clients $|Vc|$. This is unlike a case where P multicasts the content to its various clients (as in video broadcasts) which reduces the bandwidth usage by O ($|Vc|^{l}$), where $l \in [0, 1.0)$ captures the topological spread-out of the clients serviced by {P}. Though the topology model we have assumed allows content multicasts to co-exist with asynchronous content extractions by clients, we consider only the latter in our cost formulation without loss of generality. Thus, a cost model can take into account two components: the user-level QoS degradations translated into a revenue loss for the SP, the node-level resources expended by the SP to maintain the proxy replicas, and the communication resources expended for content access by various clients. Our research focuses on the QoS-oriented issues in proxy placement algorithms, by considering read-only contents.

2.4 Algorithm related issues

We treat the proxy placement problem in two steps: i) creating an overlay distribution tree $T(\{M, Vc, V'\},E')$ on the infrastructure topology $G(V, E)$ by a standard routing protocol, and ii) then indentifying the on-tree nodes $Vr \subseteq T$ to place the content proxies. Accordingly, the cost optimality in proxy placement also needs to be addressed when completing these steps.

In our research, we assume that the overlay tree is already set up under some optimality considerations. We assume that the link costs are fixed at the infrastructure topology level. Two types of tree set up algorithms are possible: a globally optimal algorithm (GL_T) and a greedy algorithm (GR_T).

A globally optimal algorithm GL_T assumes a centralized topology information and sets up a tree T ($\{M, Vc, V'\}, E'$) such that $|E'|$ is minimized. Such an algorithm is however NP-complete in complexity. Due to practical difficulties of acquiring centralized topology information, GL_T is not implemented in the standard routing protocols or their overlay variants.

The greedy algorithm GR_T embodied in DVMRP allows a client node $x \in Vc$ to set up a unicast shortest path (in the number of hops) towards the server node M. When the algorithm considers a next node $y \in (Vc - x)$ to set up a unicast path towards M, a node z in the path from x (that is already set

up) where the path from *y* meets is treated by the algorithm as the branch node to connect *x* and *y*. The algorithm proceeds until all the nodes *Vc* are connected to *M* through a distribution tree *T* ({*M, Vc, V'}, E'*). The greedy algorithm to set up trees however is sub-optimal, i.e., $|E'_{GRT}| \geq |E'_{GLT}|$. Referring to Figure 1, the GR_T algorithms finds a tree with 7 hops; whereas, the GL_T algorithm would find a tree with 6 hops (the edges *{g, f, e}* are found in the best case, as against the edges *{a, m, f, e}* found in the sub-optimal case shown in the Figure 1).

2.5 Layered vs. integrated approach for algorithm construction

There are drawbacks in employing the 2-tier (layered) approach, namely, deciding on the proxy placement *after* the overlay tree *T* has been set up. They arise due to the layered construction of algorithms: one, to construct the tree *T*, and the other, to decide on $Vr \subset T$. We state the pros and cons below (refer to Figure 1).

1. The cost considerations are different in the two layers;

   For instance, the number of hops in a path is treated as the cost in the topology layer, whereas the volume of traffic flows in the various hops is treated as the cost in the proxy management layer.

Accordingly, an optimal path at the topology level need not be optimal at the proxy replication level.

2. Greedy algorithm for tree construction GR_T is only sub-optimal;

The induced sub-optimality in the tree construction at the infrastructure topology level percolates into the proxy management layer. Accordingly, the optimality in proxy placement can only be interpreted within this context.

In contrast, an *integrated* approach can yield a better optimality by infusing the cost considerations meaningful for proxy management into the tree set up algorithm executed at the topology level. We denote the optimal algorithms developed therein as $GL_{T \oplus R}$ and $GR_{T \oplus R}$, as the case may be -- where $T \oplus R$ denotes the integration of tree construction and proxy placement functions.

With a 2-tier approach however, the proxy placement problem becomes algorithmically tractable and simpler, namely, a globally optimal algorithm GL_R requires analyzing $C_{|vr|}^{|v'|}$ distinct placements of server proxies. The cost model meaningful at the proxy management layer is employed to compute the costs of various placements and compare them.

In our present work, we study the 2-tier approach as a proof of concept in simplifying the development of proxy placement algorithms. We first evaluate two types of algorithms: GL_R that yields global optimality and GR_R that implements a variant of greedy algorithms proposed in Qiu et al. [2001]. We then provide our algorithm, GA_R, which is a genetic algorithm based determination of proxy placement. Since overlay tree setup algorithms are used in a larger time scale than the proxy placement algorithms. They are usually used by the system operator during installation of server infrastructure or while upgrading the hosting infrastructure and runs once every few months. The proxy placement algorithms are run more often, as they need to react to possibly rapidly changing situations such as flash crowds. We will study the proxy placement problem on top of a globally optimal tree setup algorithm GL_T .

3. Metrics to evaluate content distribution network

All content distribution network systems need to change proxy placements in order to maintain high performance in the mean time minimize cost. The metrics to evaluate content distribution network are

those used to evaluate the system performance. The system will react when the system performance degrade from the acceptable interval and adapt its configuration if necessary. Depending on the performance optimization criteria, a number of metrics must be carefully selected to accurately represent the performance of the system. The metrics such as client latency, network distance, network resources usage, object hosting cost can be used to evaluate the system performance [Sivasubramanian et al 2004]. The metrics related to network usage and object hosting cost are used to control the system maintenance cost, which should remain under control by the system's operator.

3.1 Client latency

Client latency can be modeled in different ways. We consider a client-server system and follow the method described in Dykes et al. [2000], modeling the total time to process a request, from the client's prospective, as $T = T_{DNS} + T_{conn} + T_{res} + T_{rest}$. T_{DNS} is the DNS lookup time to find the server's network address. Cohen and Kaplan [2001] found that DNS search time may vary greatly because of cache misses (in its cache, client's local DNS server does not have the address), in many cases it is below 500

milliseconds. T_{conn} is the TCP connection time between client and server. Zari et al. [2001] report that T_{conn} is often below 200 milliseconds, but like the DNS case, very high values up to 10 seconds may also occur. T_{res} is the time needed to transfer a request from client to server and receiving the first byte of the response. T_{res} is comparable to the round-trip time (RTT) between two nodes, but includes the time the server spent to handle the incoming request. Finally, T_{rest} is the time to complete the transfer of the entire response.

When deal with client latency, we considered two different kinds of forms. The end-to-end latency is taken as the time needed to send a request to the server, and is often taken as $0.5T_{res}$, possibly including the time T_{conn} to setup a connection. The client latency is defined as $T - T_{rest}$. This latency metric reflects the real delay observed by a user. Obtaining accurate values for latency metrics is not a trivial task. It may require specialized mechanisms, or even a complete infrastructure. Predicting client latency not only involves measuring the round-trip delay between two nodes (which is independent of the size of the response), but may also require measuring bandwidth to determine T_{rest}. The latter has shown to be particularly cumbersome requiring sophisticated techniques [Lai and Baker 1999].

3.2 Network distance

As an alternative to client latency, we can consider network distance such as number of network-level hops or even the geographical distance between two nodes. In these cases, the underlying assumption is that there exists a map of the network where the network distance metric can be expressed. Maps can have different levels of accuracy. Some maps depict the Internet as a graph of Autonomous Systems (ASes), thus unifying all machines belonging to the same AS. They are used for example by Pierre et al. [2002]. The graph of ASes is relatively simple and easy to operate on. However, because ASes significantly vary in size, this approach can suffer from inaccuracy. Other maps depict the Internet as a graph of routers, thus unifying all machines connected to the same router [Pansiot and Grad 1998]. These maps are more detailed than the AS-based ones, but are not satisfying predictors for latency. For example, Huffaker et al. [2002] found that the number of router hops is accurate in selecting the closest server in only 60% of the cases. Finally, some systems use proprietary distance calculation schemes, for example by combining the two above approaches [Rabinovich and Aggarwal 1999].Constructing and exploiting a map of the Internet is easier than running an infrastructure for latency measurements. The maps

can be derived, for example, from routing tables. McManus [1999] shows that the number of hops between ASes can be used as an indicator for client latency.

3.3 Network resources usage

Another important metric is the total amount of network resources consumption. Such resources could include routers and other network elements, but often refer to consumed bandwidth. The total network usage can be classified into two types. Internal usage is caused by the communication between proxy servers to keep them consistent. External usage is caused by communication between clients and proxy servers. Preferably, the ratio between external and internal usage is high, as internal usage can be viewed as of overhead introduced merely to keep replicas consistent. On the other hand, since clients retrieve content from proxy instead of master server so the overall network usage may decrease.

[Sivasubramanian et al 2004] modeled the problem as follow. Consider a nonreplicated document of size s bytes that is requested r times per second. The total consumed bandwidth in this case is r · s, plus the cost of r separate connections to the server. The cost of a connection can typically be

expressed as a combination of setup costs and the average distance that each packet associated with that connection needs to travel. Assume that the distance is measured in the number of hops (which is reasonable when considering network usage). In that case, if l_r is the average length of a connection, we can also express the total consumed bandwidth for reading the document as $r \cdot s \cdot l_r$. On the other hand, suppose the document is updated w times per second and that updates are always immediately pushed to, say, n replicas. If the average path length for a connection from the server to a replica is l_w, update costs are $w \cdot s \cdot l_w$. However, the average path length of a connection for reading a document will now be lower in comparison to the nonreplicated case. If we assume that $l_r = l_w$, the total network usage may change by a factor w/r in comparison to the nonreplicated case.

3.4 Object hosting cost

Another obvious but important metric is object hosting cost. Obviously the bigger size of the object we replicate to proxy server, the

more cost we will have. Also, more proxy server we deploy to replicate the object, the more we pay for resources like server power and storage. Given a proxy placement X, object hosting cost f(X) has positive correlation to the number of proxy servers.

4. Objective function for optimizing proxy placement

When we optimize proxy placement in content distribution network we want to maximize the user experience in the mean time to minimize the resource consumption. These two objects are conflict with each other. To maximize the user experience we need to place proxy as many as possible, but that will increase the resource consumption. To minimize the resource consumption we need to minimize the number of proxy placement and that will increase the user latency so the user experience will decrease. We are dealing with a multi-objective optimization problem here. By optimize a single objective function we obtain a partial view of the results. To solve this problem we can use traditional methods in multi-objective optimization theory. One of them is weighted sum method. We can assign a weight to each of the objective functions to create a single objective model. F(X) =

$w*C^{\alpha}(X) + (1-w)* f^{\beta}(X)$ where X is the proxy placement, C(X) is the objective function for user experience, f(X) is the objective function for resource consumption. An analysis of this type of solution shows that function F(X) obtained from the sum is really a combination of the function C(X) and function f(X). In our research we use the number of proxies to represent the resource consumption, in the real world, the resource consumption may not necessary be a linear function of the number of proxies. Below we will focus on the objective function for user experience.

For simplicity we model user experience as the total network transfer cost. We only consider content movement, because the control message overhead can be neglected compare to the overall cost of content movement.

There are I servers in the network. $Server_i$, i = 1, 2, …, I. and has K clients that request objects at (aggregate) rate r_k where k = 1, 2, … , K. We have the following variables: $x_i = 1$ if proxy has been placed at server i, $x_i = 0$ otherwise. The constrain is that only J servers are allowed to place proxies.

The goal is to choose the x_i's so that a given performance metric is minimized. For simplicity, our goal is to minimize the average number of hops that a request must traverse. This reflects the download time of an object to some degree and can be used as an indicator of the user perceived latency.

We denote the vector of all x_i's by **X**. Furthermore, we assume that each object j is initially placed on an origin server; we denote by O_j. We assume that all of the objects are always available in their original servers, regardless of the placement **X**. We denote the placement of objects to origin servers as $\mathbf{X}_o$.

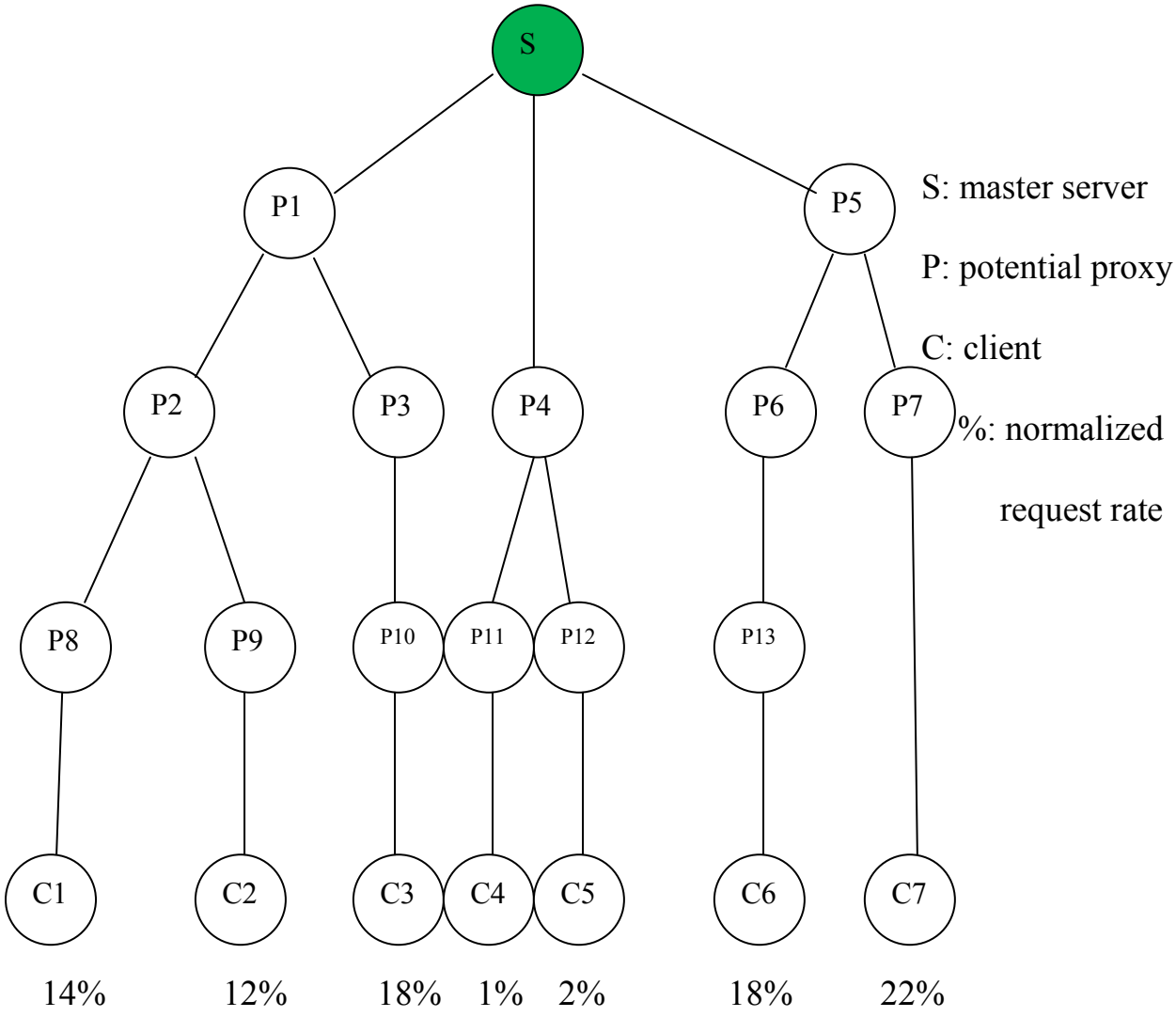
The number of hops that a request must traverse from client k is $d_k(\mathbf{X})$ where $d_k(\mathbf{X})$ is the shortest distance to a copy of a proxy from client k under the placement **X**. This nearest copy is either in the origin server O or in another proxy server where the object has been replicated. We assume that the client is always redirected to the nearest copy.

Let $R = \sum_{k=1}^{K} r_k$ be the total request rate of all clients. The average number of hops from all clients is then

$$C(\mathbf{X}) = \frac{1}{R} \sum_{k=1}^{K} r_k C_k(\mathbf{X}) = \frac{1}{R} \sum_{k=1}^{K} r_k d_k(\mathbf{X}) = \sum_{k=1}^{K} s_k d_k(\mathbf{X})$$

Where $s_k = r_k / R$.

This cost function represents the long term average cost. For a large number of clients and potential proxy sites, it is not feasible to solve this problem by exhaustive search. See figure 2.



Choose 4 sites from 13 potential nodes: 13*12*11*10/4*3*2*1possibilities

Choose m sites from n potential nodes: n! / (m!*(n-m)!)

Figure 2: Exhaustive search proxies for CDN

5. Existing Proxy Placement Algorithms

Proxy placement forms a controllable input parameter of the objective function. Changes in uncontrollable parameters, such as client request rate or client latencies, may warrant changing the proxy locations. In such case, the adaptation-triggering component triggers the proxy placement algorithms, which subsequently adapt the current placement to new conditions [Sivasubramanian et al 2003].

Ravindran and Wu [2005][2006] described a protocol-level adaptation mechanism to enhance the performance of network service provisioning to clientele. In their model, a service provider (SP) maintains multiple protocol modules to exercise the infrastructure resources. Each protocol exhibits a certain degree of cost optimality (in terms of resource usage) in distinct operating regions of the network infrastructure. This is related to the dissertation topic. For example, when place proxy during system set up, due to the sufficient time provided, the SP could select exhaustive search to maximize the service performance under the resource-level operating conditions. When the client access pattern suddenly changed, due to the immediate requirement of the change of proxy placement, the SP may switch to better performance heuristic search method such as genetic algorithm to response quickly. Their model allows a 'dynamic switching'

from one protocol module to another at run-time based on the changing network conditions. The overall goal is to offer a sustained access to the service with a resource-optimal and QoS-compliant service offering. They describes 'protocol switching' as an architectural foundation for building cost-effective network services.

Rabby, Ravindran and Wu [2010] considered a wide-area video conferencing application where the video sources can adapt their send rates according to the available bandwidth in the network paths. They advocated a joint rate control of the sources to relieve the congestion, instead of running multiple instances of a single source adaptation algorithm and additively superposing their results. The existing techniques work nicely with single-source trees, but do not work optimally in the case of multi-source trees with shared QoS goals. Using the well-known AIMD-based adaptation procedures, they incorporate the topology inference mechanism into a coordinated rate adaptation algorithm executed by the loss-experiencing sources. They provided a simulation based evaluation of their algorithm to corroborate the benefits. Even though the topic is on rate-adaptive video multicast protocols, there are some heuristics-based optimization elements described (to split the send rate reductions among multiple sources to relieve

link congestion) and that is related to our optimization algorithm on content distribution networks.

Lowe et al. [1999] propose a placement algorithm based on the assumption that the underlying network topologies are trees and solve it using dynamic programming techniques. The algorithm is designed for web proxy placement. The algorithm works by dividing a tree T into smaller sub-trees T_i; the authors show that the best way to place t proxies is by placing T_i proxies for each T_i such that $\Sigma\ T_i = t$. The algorithm is shown to be optimal if the underlying network topology is a tree. This nature of problem formulation is more relevant for content placement, where every document has a single origin web server.

Qiu et al. [2001] propose a greedy algorithm. In each iteration, the algorithm selects one server, which offers the least cost, where cost is defined as the average distance between the server and its clients. In the ith iteration, the algorithm evaluates the cost of hosting a proxy at the remaining $N - i + 1$ potential sites in the presence of already selected i - 1 servers. The computational cost of the algorithm is $O(N^2K)$. The authors also present a hot-spot algorithm, in which the replicas are placed close to the clients generating most requests. The computational complexity of the hot-spot

algorithm is $N^2 + min\ (NlogN, NK)$. The authors evaluate the performance of these two algorithms and compare each one with the algorithm proposed in

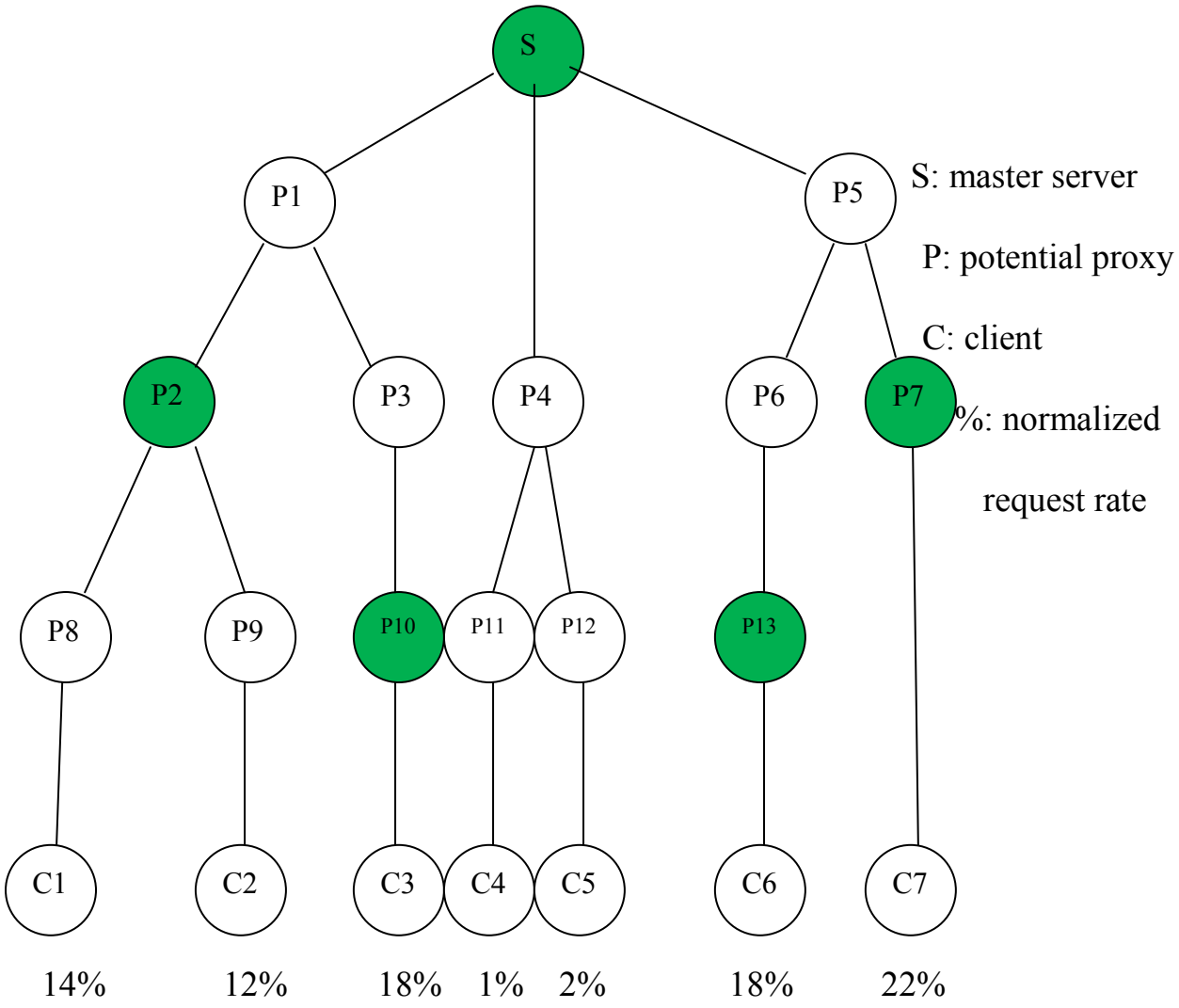


Figure3: Greedy algorithm successfully finds the optimal solution

Lowe et al. [1999]. Their analysis shows that the greedy algorithm performs better than the other two algorithms and its performance is only 1.1 to 1.5

times worse than the optimal solution. The authors note that the placement algorithms need to incorporate the client topology information and access pattern information, such as client end-to-end distances and request rates. The drawback of these algorithms is that it cannot guarantee to get the global optimum. It is easy to trap in the local optimum. See Figure 4.

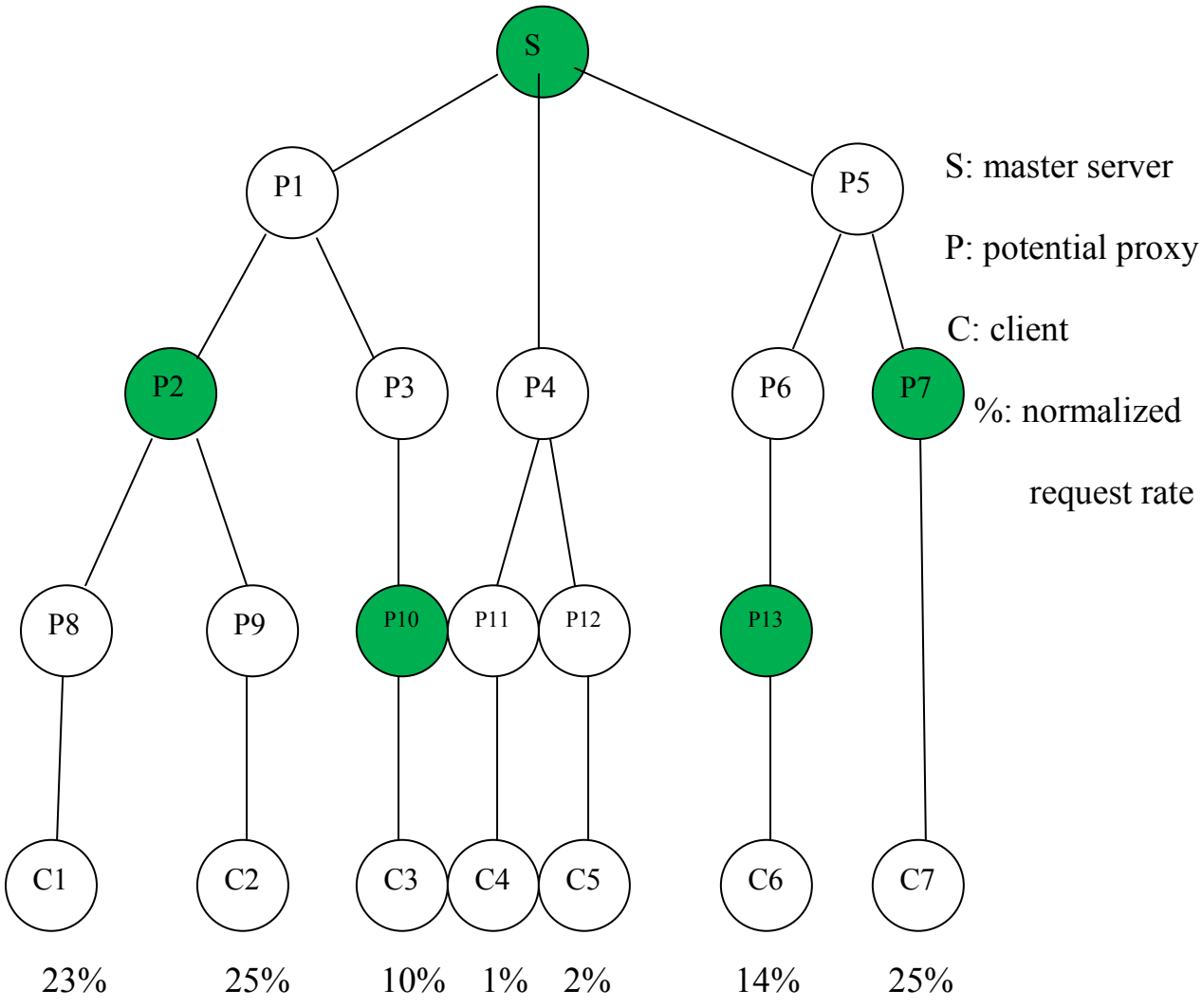


Figure 4: Greedy algorithm finds the sub-optimal solution

Radoslavov et al. [2001] propose two proxy server placement algorithms that do not require the knowledge of client location but decide on proxy location based on the network topology alone. The proposed algorithms are *max router fanout* and *max AS/max router fanout*. The first algorithm selects proxy servers closest to the router having maximum fanout in the network. The second algorithm first selects the Autonomous System (AS) with the highest fanout, and then selects a server within that AS that is closest to the router having maximum fanout. The performance studies show that the second algorithm performs only 1.1 to 1.2 times worse than that of the greedy algorithm proposed in Qiu et al. [2001]. Based on this, the authors argue that the need for knowledge of client locations is not essential. However, it must be noted that these topology-aware algorithms assume that the clients are uniformly spread throughout the network, which may not be true. If clients are not spread uniformly throughout the network, then the algorithm can select proxy servers that are close to routers with highest fanout but distant from most of the clients, resulting in poor client-perceived performance.

In Radar [Rabinovich and Aggarwal 1999], every host runs the proxy placement algorithm, which defines two client request rate thresholds: R_{rep} for proxy replication, and R_{del} for object deletion, where $R_{del} < R_{rep}$. A

document is deleted if its client request rate drops below R_{del}. The document is replicated if its client request rate exceeds R_{rep}. For request rates falling between R_{del} and R_{rep}, documents are migrated to a proxy server located closer to clients that issue more than a half of requests. The distance is calculated using a Radar-specific metric called preference paths. These preference paths are computed by the servers based on information periodically extracted from routers.

In SPREAD [Rodriguez and Sibal 2000], the proxy servers periodically calculate the expected number of requests for an object. Servers decide to create a local proxy if the number of requests exceeds a certain threshold. These servers remove a proxy if the popularity of the object decreases. If required, the total number of replications of an object can be restricted by the object owner.

Kangasharju et al. [2001a] model the proxy placement problem as an optimization problem. The problem is to place K objects in some of N servers, in an effort to minimize the average number of inter-AS hops a request must traverse to be serviced, meeting the storage constraints of each server. The problem is shown to be NP-complete and three heuristics are proposed to address this problem. The first heuristic uses popularity of an object as the only criterion and every server decides upon the objects it

needs to host based on the objects' popularity. The second heuristic uses a cost function defined as a product of object popularity and distance of server from origin server. In this heuristic, each server selects the objects to host as the ones with high cost. The intuition behind this heuristic is that each server hosts objects that are highly popular and also that are far away from their origin server, in an effort to minimize the client latency. This heuristic always tries to minimize the distance of a proxy from its origin server oblivious of the presence of other replicas. The third heuristic overcomes this limitation and uses a coordinated replication strategy where proxy locations are decided in a global/coordinated fashion for all objects. This heuristic uses a cost function that is a product of total request rate for a server, popularity and shortest distance of a server to a copy of object. The central server selects the object and proxy pairs that yield the best cost at every iteration and re-computes the shortest distance between servers for each object. Using simulations, the authors show that the global heuristic outperforms the other two heuristics. The drawback is its high computational complexity.

Karlsson et al.[2002] compare the benefits of caching with replica placement algorithms (RPA) when used in content delivery networks (CDN). First, they identify what RPAs provide the best client perceived

latency improvements. They find that taking object accesses into account usually makes a big difference. They find that the decentralized popularity algorithm and the storage constrained, multi-object Greedy algorithm overall work the best across a number of performance metrics. Popularity is generally preferable as it is much less computationally expensive. When they compare caching to Popularity and the above Greedy algorithm, they find that when there is plenty of storage, caching is as good, if not better, than the best RPA. However, caching performs much worse than the RPAs when there is a limited amount of storage. This is mainly due to the fact that RPAs are only evaluated infrequently, e.g. once a day, and thus it is possible to communicate information about where a node can fetch from the closest replica of an object. As caching changes its stored content frequently, this is not possible for a local cache, and it has to go to the origin server that is usually located at quite some distance. However, by only evaluating the caching algorithm as infrequently as the RPAs, they can communicate the knowledge about the closest copy of an object in the same way as with an RPA. In this way, simple caching generally performs better than an RPA even for storage constrained systems.

Bartolini et al. [2003] provided a framework for the design of replica allocation schemes dynamically placing and removing replicas in response

to changing users demand. By assuming the users requests dynamics to obey to a Markovian model they have first formulated the dynamic replica placement problem as a Markovian decision process. This allowed them to identify an optimal policy for dynamic replica placement that can be used as a benchmark for heuristics evaluation and provides insights on how allocation and deallocation should be performed. Based on the findings obtained through the analytical model we derived and evaluated a centralized heuristic which allocates and deallocates replicas to reflect the requests traffic dynamics, the costs of adding, deleting and maintaining replicas, the servers load and storage limits, and the requirements on the maximum distance of the users from the 'best replica'. The heuristic performance evaluation has shown that the heuristic behavior is very close to that of the optimal placement strategy, and that the heuristic results in good performance in terms of low average number of replicas, low user-replica average distance and low number of requests that cannot be served.

Jia et al. [2003] discussed two cases of optimal data replication at transparent proxies: the proxies with unlimited storage capacities and those with limited capacities. For the former case, they gave detailed analysis of optimal data replication and proposed an efficient algorithm that computes the optimal result. For the latter, they proposed two heuristics. Extensive

simulations have been conducted for performance analysis of the data replication algorithms under various network situations. The effectiveness of transparent caching depends on the stability of the routing methods. If the routing is stable (i.e. packets from a source to a destination tend to take the same route over a long period of time), the routes used by clients to access the Web server could be modeled as a tree rooted from the server. In this case, the placement of proxies and the replication of data at proxies according to the clients' access patterns would certainly increase the hit-ratio of proxies. The Internet routing is usually stable. It was found that 80 percent of routes changes at a frequency lower than once a day. The placement of transparent proxies in the routers requires static configuration work. Once a proxy is configured, it is expected the proxy would stay with the router for a relatively long period of time. This indicates that the client response time and the reduction of web traffic can take great benefit from the proper locations of the proxies and the optimal replication of data objects. It should also be pointed out that the performance of replication method heavily depends on the accuracy of past statistical data, such as read and update frequencies of objects. Enough statistical data on client access patterns must be collected before using the data replication method. Even during the time when the replication method is in use, collecting statistical

data is also important to help adjust and improve data replication decisions from time to time.

Presti et al. [2005] Stated that the content delivery networks (CDN) paradigm is based on the idea to transparently move third-party content closer to the users. More specifically, content is replicated on CDN servers which are located close to the final users, and user requests are redirected to the "best" replica (e.g. the closest) in a transparent way, so that users perceive a better content access service. In their paper they address the problem of dynamic replica placement. Being dynamic, their solutions adoptively select the number of replicas for each content and the replicas positions to account for traffic requests dynamics. The schemes they propose are designed to minimize the overall cost paid by the CDN provider (for replicas placement, removal, and maintenance) without degrading the quality of the users perceived access service. They introduce a centralized and distributed scheme for replica placement in a dynamic traffic scenario. Then, by means of a simulation based performance evaluation, they assess the effectiveness of the proposed schemes, and compare their performance with static solutions which have been proven to perform well in the literature. Simulation results show that both the two proposed algorithms achieve very good performance, resulting in a significant improvement over

the static solutions. Despite relying on local information only, the distributed scheme has comparable performance to the centralized one. Both the two schemes result in low average distance between the users and their serving replicas, in low average number of replicas, in infrequent replicas add and tear down, and in high probability of being able to serve a request.

Tang et al. [2004] have investigated the minimal cost replica placement problem for QoS-aware content distribution. The problem has been formulated under two classes of service models (replica aware service and replica-blind service) and three different cost models (storage cost, update cost, and their combination). In replica-aware services, the content distribution system is modeled as a general graph. The associated replica placement problems are proven to be NP-complete. Several heuristic algorithms have been proposed and experimentally evaluated against a super-optimal bound obtained from the relaxed linear program. The results show that the proposed heuristics perform close to the super-optimal bound. In replica-blind services, the delivery paths with respect to a given origin server are represented by a tree topology. It is shown that the optimal solution to the associated replica placement problem for minimal update cost can be computed with a time complexity linear to the number of servers. There also exist polynomial optimal solutions to the associated replica

placement problems for minimal storage and combined costs. Dynamic programming algorithms with time complexities square to the number of servers have been proposed for these two problems

Varadarajan et al. [2003] illustrated the various necessary factors and constraints that need to be taken into account for a good placement of proxies over the Internet which reflect real world scenario more accurately and which they claim hitherto has not been completely addressed. They introduced a novel concept of host coverage characterizing every Autonomous Systems (AS) and use this stable, coarse grained measure as a long-term estimate of the load being serviced by the proxy system. They validated its applicability through an Internet study. They then posed an optimal formulation of the proxy placement problem taking into consideration all the relevant factors. They proposed a couple of proxy placement algorithms that solve the problem and analyze their behavior. In the greedy coverage approach, they deal with the host coverage dimension first and then deal with the distances. They first partition the ASs into zones in such a manner so as to equalize the total host coverage of each zone. In each zone, the way they pick ASs is similar to any spanning tree algorithm. Next they placed a proxy in each of the zones so that the cost is minimized in each of the zone.

Yang et al. [2003] analyzed the assumptions underlying the existing models for the problem of replica placement in content distribution networks. To accommodate the new characteristics brought by the multimedia applications they propose a new model for replica placement. They perform a theoretical analysis of the cost of distributing multimedia files over CDNs and find out that, contrary to the intuition, deploying as many replicas as possible is not always a good strategy. They then propose several replica placement algorithms that can determine the optimal number of replicas that should select from a given set of potential sites. By simulation they demonstrate that the performance of clients may degrade if they choose too many sites for replica placement.

Caching and content delivery are important for content-intensive publish/subscribe applications. Chen et al. [2003] proposed several content distribution approaches that combine match-based pushing and access-based caching, based on users' subscription information and access patterns. To study the performance of the proposed approaches, they built a simulator and developed a workload to mimic the content and access dynamics of a busy news site. Using a purely access-based caching approach as the baseline, their best approaches yield over 50% and 130% relative gains for two request traces in terms of the hit ratio in local caches, while keeping the

traffic overhead comparable. Even when the subscription information is assumed not to reflect users' accesses perfectly, their best approaches still have about 40% and 90% relative improvement for the two traces. Their work is the first effort to investigate content distribution under the publish/subscribe paradigm.

Alguliev et al. proposed a new model of optimum placement of servers and Web contents in a Content Delivery Network that is intended to minimize the cost of delivery of content to the ultimate users is proposed. The model also takes into account the structure of the network and the weight of each web content in the network nodes. A mathematical formulation of the proposed model reduces to a problem of linear integer programming. In the study synthesis of a neural network for the solution of a problem of linear integer programming is also described.

Adaptive content networking is a promising new approach aimed at scalable delivery of content to a pervasive client population. By adaptive content delivery networks (A-CDN) content is adapted, replicated and delivered to the clients in a cost-quality-optimized fashion. The integration of content adaptation into CDNs minimizes the interference of adaptation with replication effectiveness Buchholz et al. [2004] studied replica placement in Adaptive Content Distribution Networks. In A-CDNs, different

representations of an object may be stored in the surrogates and adapted to the clients by the CDN during delivery. Hence, RP (Replication Placement) in A-CDNs is targeted at deciding which representation of which object to store in which surrogate. They have introduced a formal model for cost-quality-optimized adaptive content networking and defined the RP optimization problem. The definition of the objective function assumes the optimal adaptation path to satisfy a request to be known. Hence, optimal adaptation path composition is an inherent sub-problem of RP. They have proposed an algorithm to tackle this problem by mapping it onto conventional shortest path search. The actual RP optimization problem is NP-hard. That is why they favor heuristic approaches. They have presented plain and a greedy ranking algorithm. Ranking approaches have proven successful in similar problems such as general RP or view selection in data warehouses.

The architecture of overlay networks should support high-performance and high-scalability at low costs. This becomes more crucial when communication, storage costs as well as service latencies grow with the exploding amounts of data exchanged and with the size and span of the overlay network. For that end, Unger et al. [2004] proposed multicast methodologies can be used to deliver content from regional servers to end

users, as well as for the timely and economical synchronization of content among the distributed servers. Another important architectural problem is the efficient allocation of objects to servers to minimize storage, delivery and update costs. They suggest a multicast based architecture and address the optimal allocation and replication of dynamic objects that are both consumed and updated. Their model network includes consumers which are served using multicast or unicast transmissions and media sources (that may be also consumers) which update the objects using multicast communication. General costs are associated with distribution (download) and update traffic as well as the storage of objects in the servers. Optimal object allocation algorithms for tree networks are presented with complexities of O(N) and O(N2) in case of multicast and unicast distribution respectively.

Cahill et al. [2005] outlined a scalable and efficient algorithm for the placement of video content in a Video CDN (VCDN). The algorithm makes replication decisions based on resource costs (currently: link cost and storage cost). When a client joins a cluster, the proxy which is currently serving the cluster initiates a cost evaluation procedure, whereby the proxy checks to see if any of its existing clusters (viewing a particular object) would be better served by a different proxy. The evaluation procedure will

determine if the object should be replicated and if so, where to replicate the object. The VCDN algorithm also includes a parameter which is used to delay the evaluation procedure which would be executed when a client joins. It is hoped that by delaying the evaluation by a predetermined amount of time, the placement algorithm will not make replication decisions that may be adversely affected by the *channel-hopping* effect. They evaluate the effectiveness of the placement algorithm, detailing the important parameters associated with the algorithm. The VCDN placement algorithm is also compared with the well known closest-proxy algorithm to evaluate its effectiveness. They conclude that an efficient placement algorithm should include a small evaluation delay to ignore channel-hoppers, while also looking at placing a single replica to serve multiple clusters.

Bhulai et al. explored the design of streaming CDNs to attain (close to) minimum delivery costs. This is achieved by developing an optimization model, which includes the network bandwidth required for each hop in a delivery tree, to determine the optimal replica placement and the delivery trees for multiple media files. Our model can deal with different scalable streaming protocols, including periodic broadcast protocols and hierarchical multicast streaming merging protocols. They introduced fast and accurate heuristic algorithms for solving the optimization model so that it can be

applied to large streaming CDNs. Moreover, they discussed the trade-off between the start-up delay and the network bandwidth, and the trade-off between the number of replica servers and the network bandwidth. The results show that the heuristics are well-suited to be used in the design and optimization of commercial streaming CDN infrastructures.

Xu et al. [2005] proved that the QoS-Aware object replication placement problem in content distribution network is a NP-complete problem and an optimal solution is not feasible. To solve this problem, they proposed a three-stage mechanism to achieve the best performance while meeting the QoS requirements of different clients. They defined a minimal object replication set (MORS) to represent the minimal system resource requirement for certain QoS guarantee. After a MORS is generated, more objects are replicated on the spare server space. A combination of three heuristic algorithms: random, popularity and greedy are introduced to achieve the near-optimal performance. They did extensive experiments to evaluate the performance of different algorithm combinations and compare with the super optimal solution. They observed that it is important to take client access behavior into account when the system make the object replica deployment decision. The neglect of this information will cause at least 30% to 40% system performance degradation.

In DHT-based P2P systems (Distributed Hash Table-based Peer to Peer systems), Replication-based content distribution and load balancing strategies consists of such decisions as which files should be replicated, how many replicas should be created and where to replicate them in order increase the system performance in the presence of non-uniform data and access distribution. There are many works on replica placement policies; however, the impact of system workload on different replica placement strategies is not well studied. Alqaralleh et al. investigated this problem under the context of content addressable overlay networks. They compare a trace based replica placement algorithm with two of its variations, namely random placement and priority based placement under different workloads. Their experimental results show that the effect of replica placement policy is highly affected by the workload of the system, which indicates that an adaptive replica placement strategy is desirable for content distribution in an overlay network. The three algorithms in pseudo-code are as follows:

Algorithm 1: Content node creation - CDNQueryStat

function New_Content_Node ( )

if new content node creating flag is not set

set new content node creating flag

find the node *n* from *Query_Stat* table

with maximum query rate, assume *n*

passes most queries to key *k* and $n \notin cdnk$

if not found

randomly select node *n* from the DHT ID space, $k\ n \notin cdn$

assign *n* the most requested Key in *Query_Stat table*

endif

send *cdn_request* message to *n*

endif

Algorithm 2: Content node creation - CDN-Rand

function New_Content_Node( )

if new content node creating flag is not set

set new content node creating flag

randomly select node *n* from the DHT ID space, $n \notin cdn$

assign *n* the most requested Key in *Query_Stat table*

send *cdn_request* message to *n*

endif

Algorithm 3: Content node creation - CDN-PR

function New_Content_Node ( )

if new content node creating flag is not set

set new content node creating flag

find the node **n** from cdnA, where n ∈cdnA , and **load n** < Wn – m and assign n the next most requested Key in Query_Stat table

if not found

randomly select node n from the DHT ID space, n∉cdn

assign n the most requested Key in Query_Stat table

endif

send cdn_request message to n

endif

Guo et al. [2007] after formulated The RAPP(Recursive Approach for Proxy Placement) problem**:** Given a complete directed graph $G = (V, E)$ of N nodes, find a constrained directed spanning tree T of G rooted at node r, such that $L($denote the average end-to-end delay) is minimized, and T is subject to constraints on: 1) K, $K < N$, nodes (including node r) are selected

as internal nodes, all other $N - K$ nodes are for leaf nodes; 2) Lmax is bounded by LB max; 3) F is restricted to FLB. They stated their experiments have demonstrated that the K-median approach does not yield good quality proxy placement solutions from the source-to-end latency performance point of view in overlay multicast networks. Strongly motivated, they have proposed in their paper a new approach and formulated the RAPP problem to address inter-domain proxy placement, aiming to support large-scale Internet live media streaming more efficiently using overlay multicast mechanisms. They have presented two efficient heuristic methods for the RAPP problem, which have been shown by simulation experiments to be scalable for large size overlay multicast networks and yield near-optimal latency performance of overlay skeleton trees. Simulation experiments have confirmed that their proposed algorithms can significantly improve the latency performance of overlay skeleton trees in comparison with the K-median approach. In addition, the performance gain is not much sensitive to network dynamics.

A forward cache is an HTTP cache deployed within an Internet Service Provider's (ISP) network caching all cacheable HTTP traffic accessed by the customers of the ISP. In contrast to CDNs a forward cache is deployed for the customers benefit and under the control of the ISP, rather

than for the benefit of the content owner. To reduce the costs associated with forward caching in an ISP Erman et al. [2009] proposed Network-Aware Forward Caching, their goal is to find the set of addresses at each POP (Point of Presences) that when cached maximizes the cost savings for the network. A key advantage of this approach is that they can reduce the network costs associated with forward caching to maximize the benefit obtained from their deployment. They show in their simulation that a 37% increase to net benefits could be achieved over the standard method of full cache deployment to cache all POPs traffic. In addition, they show that this maximal point occurs when only 68% of the total traffic is cached.

Triantafillou et al. [2010] designed and implemented ProxyTeller, a tool to guide the placement of proxies and/or server replicas of origin servers in a content delivery network. ProxyTeller accepts as inputs the network topology, the desired sizes of proxy caches, the characteristics of the workload of the community served by a proxy, and the desired performance improvement. The output of ProxyTeller is the position and the number of proxies required in order to achieve the stated performance goals in terms of access latency, network bandwidth, server load, and storage requirements. ProxyTeller can be used for both forward and reverse proxies and it will be of real use to the organizations maintaining them, and in particular: to

ISPs/CDNs, to optimally decide on required capital investments, weighing them against expected performance improvements, and to content providers, which will be able to estimate their origin server off-load possible (as this is depicted in cache hit ratios) from a potential contract with a CDN and the expected performance improvements seen by the consumers of their content.

Wu et al. [2009] stated when design a CDN we need to find proxy server placement to provide its clients with the best available performance while consuming as little resource as possible. This is an optimization problem. Among the solutions greedy algorithm yields better result with low computational cost. The drawback is that it is easy to trap in the local optimum. We propose genetic algorithm to solve this problem. They mathematically model the optimization problem and then give details about how to apply genetic algorithm to proxy server placement problem. Simulation results for a simple topology are presented for both greedy algorithm and genetic algorithm. They only worked with small topologies, large real world size topology need to be tested for their algorithm and simulation.

6. Using genetic algorithm to solve the proxy placement problem

A genetic algorithm (or short GA) is a search algorithm used in computing to find true or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are inspired by biological evolution such as inheritance, crossover, mutation, and selection.

Why use genetic algorithm in proxy placement problem? The most important reason is that genetic algorithm is parallel. Other algorithms such as greedy algorithm are serial and can only explore the proxy placement solution space to a problem in one direction at a time, the solution they discover are easy to be trapped in local optimal. Since GA have multiple offspring, they can explore the proxy placement solution space in multiple directions at once, giving them a greater chance each run of finding the optimal proxy server location.

Due to the parallelism that allows genetic algorithm to evaluate many proxy locations at once, genetic algorithm is particularly well suited to solving proxy placement problems where the space of all potential

combination of proxy server location is truly huge - too vast to search exhaustively in reasonable amount of time. The parallelism of a GA allows it to succeed in even this enormous number of possibilities, successfully finding optimal or very good results in a short period of time, it only need to sample small part of the large possibilities of proxy server locations.

Another advantage of genetic algorithm is that it performs well in proxy placement problem where the fitness landscape is complex and the objective function has many local optima. Greedy algorithm can become trapped by local optima. Genetic algorithms have proven to be effective at escaping local optima and discovering the global optimum in a very complex fitness landscape. Although in reality there is usually no way to tell whether a given solution to the proxy server placement problem is the one global optimum or just a very high local optimum, it can almost always deliver at least a very good solution. GA's major components - parallelism, selection, mutation, and crossover work together to accomplish this. In the beginning, the GA generates a diverse initial population (diverse locations of proxy servers), casting a "net" over all the potential proxy server locations. Small mutations enable each proxy server location to explore its immediate neighborhood, while selection focuses progress, guiding the algorithm's offspring to better locations. However, crossover is the key element that

distinguishes genetic algorithms from other algorithms, without crossover, each individual solution is on its own, exploring the search space in its immediate neighbor without reference to what other solution may have discovered. With crossover in place, there is a transfer of information between successful placement of proxy server locations – individual placements can benefit from what others have learned, they can be mixed and combined, with the potential to produce an offspring placement that has the strengths of both its parents and the weaknesses of neither.

GA knows nothing about the proxy placement problem it is deployed to solve. Instead of using previously known topology information to guide each step and making changes with a specific eye towards improvement, as human designers do, it make random changes of the placement to its candidate solutions and then use the objective function to determine whether those changes produce an improvement. The virtue of this technique is that it allows genetic algorithms to start out with an open mind. Since its decisions are based on randomness, all possible search pathways are theoretically open to a GAs; by contrast, any problem-solving strategy that relies on prior knowledge must inevitably begin by ruling out many pathways, therefore missing any novel solutions that may exist there, GAs do not have this problem. Similarly, any technique that relies on prior knowledge will break

down when such knowledge is not available, but again, GAs are not adversely affected by ignorance. Through their components of parallelism, crossover and mutation, they can range widely over the fitness landscape, exploring regions which intelligently produced algorithms might have overlooked, and potentially uncovering solutions that might never have occurred to other algorithms.

Although genetic algorithm has so many advantages when applying to the optimization in proxy placement in content distribution networks comparing with greedy algorithm, it has increased computational cost so it is slower in most cases. Especially when the network size is large, the encoding chromosome is long and the population is bigger, genetic algorithm needs more iterations than greed algorithm.

When we apply the genetic algorithm in the real world content distribution networks the time issue is even more important. Since the content distribution network is complex by its nature we do not have a mathematical model for it. So, we cannot make a conclusion if a possible solution is good or not based on the simplified topology model. Given a proxy placement in content distribution network, we do not have a mathematical formula to calculate the cost. We have to gather information

from performance monitoring agents. Such activities are time-consuming. Here, greedy algorithms may be a better choice. So, we have to be careful about choosing the cases where genetic algorithms can be applied. When the response time is considered more important we may apply greedy algorithm, when the system performance is considered more important we may apply genetic algorithm.

We will demonstrate GA's design in detail, by presenting our encoding mechanism and then the selection, crossover, mutation and termination operators.

The genetic algorithm requires that any trial solution of the proxy placement be represented by a coded string of certain length similar to the structure of a chromosome. Here each bit of the string is the numerical value of the label of each proxy.

Fitness value f for proxy placement problem: $\sum_{i=1}^{K} r_k d_k (X)$ where r_k is the request rate of client k, d_k **(X)** is the shortest distance to a copy of a proxy from client k under the placement X.

6.1 Pseudo-code

- Choose initial population (proxy server placements)
- Evaluate the fitness of individuals in the population

Repeat

- Select best-ranking individuals to reproduce
- Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
- Evaluate the individual fitness of the offspring
- Replace worst ranked part of population with offspring

Until terminating condition

6.2 Initialization

In the repetitive procedure of the genetic algorithm, the first step in the functioning of a genetic algorithm is the generation of an initial

population. Each member of this population encodes a possible solution to the proxy placement. After creating the initial population, each individual is evaluated and assigned a fitness value according to the fitness function. It has been recognized that if the initial population to the GA is good, then the algorithm has a better possibility of finding a good solution and that, if the initial supply of building blocks is not large enough or good enough, then it would be difficult for the algorithm to find a good solution. In our experiment, the initial population will be randomly generated. Each string is an array of randomly generated numerical label of each proxy. The length of the array equals the number of proxies chosen. After initialization is the recombination and mutation.

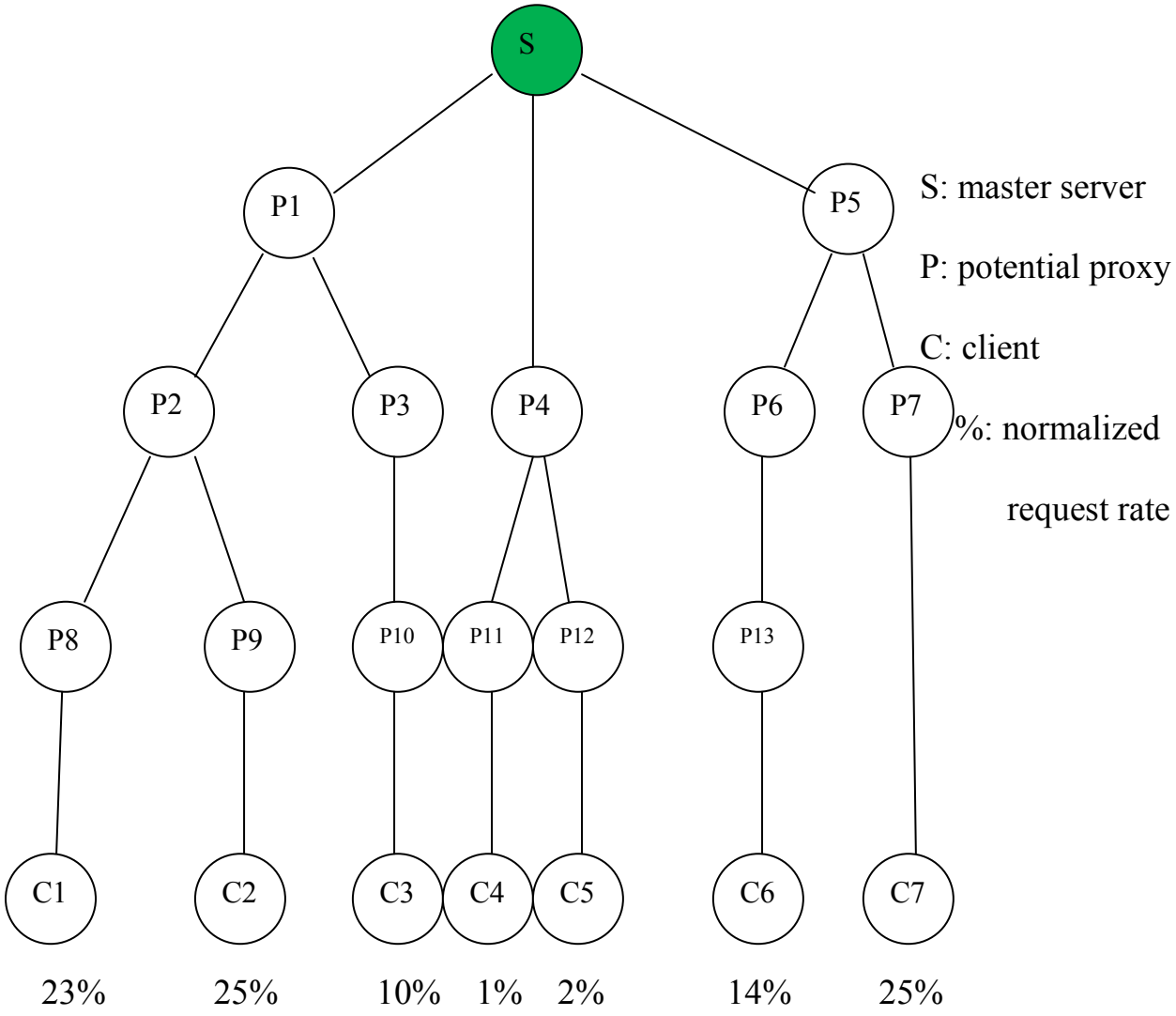


Figure 5: Initial population of GA

Random seeds: 2-6-8-9, 2-7-9-11, 3-5-7-12 …

6.3 Reproduction

The individual proxy placements in the population are evaluated, and their fitness values are calculated. Based on these fitness values, individual

proxy placements are now selected to contribute their genetic information to the next generation. The idea is that individuals with a high fitness, compared to the population as a whole, will have a high chance of being selected, and individuals with a low fitness value will have a low chance. Inspired by biology, two offspring are formed via partial exchange of bits between two selected parents by using a crossover operator. Crossover occurs with some specified probability of crossover for each pair of parents selected in the previous step. Each offspring contains part of the proxy sites from one parent and part of proxy sites from the other parent. The crossover point is chosen at random. Figure 6 provides an overview of the crossover operator.

According to the crossover operator, parents could have the same proxy location thus a child could inherit them both, resulting in a redundant proxy location in the offspring. It seems this is a problem, but we really do not need to worry about that because a proxy placement with redundant location will have lower fitness value hence the offspring will have lower possibility to be select to reproduce new offspring.

We use one-point crossover. A crossover point on the parent array is selected. All data beyond that point is swapped between the two parent arrays. Mutation is performed by simply flipping every array element with a

certain probability u, known as the mutation rate. Although mutation is not the primary search operation and some algorithms omit it, it is very useful in our design. The reason is that in order for our algorithm to have practical applications it should achieve good performance dealing with very large chromosomes. A one point-crossover alone would not be able to explore the solution space fast while a multiple point crossover would result in highly disruptive.

Selection is stochastic designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large.

6.4 Termination

This generational process is repeated until a termination condition has been reached. The termination condition here is the highest-ranking solution's fitness has reached a plateau such that successive iterations no longer produce better results.

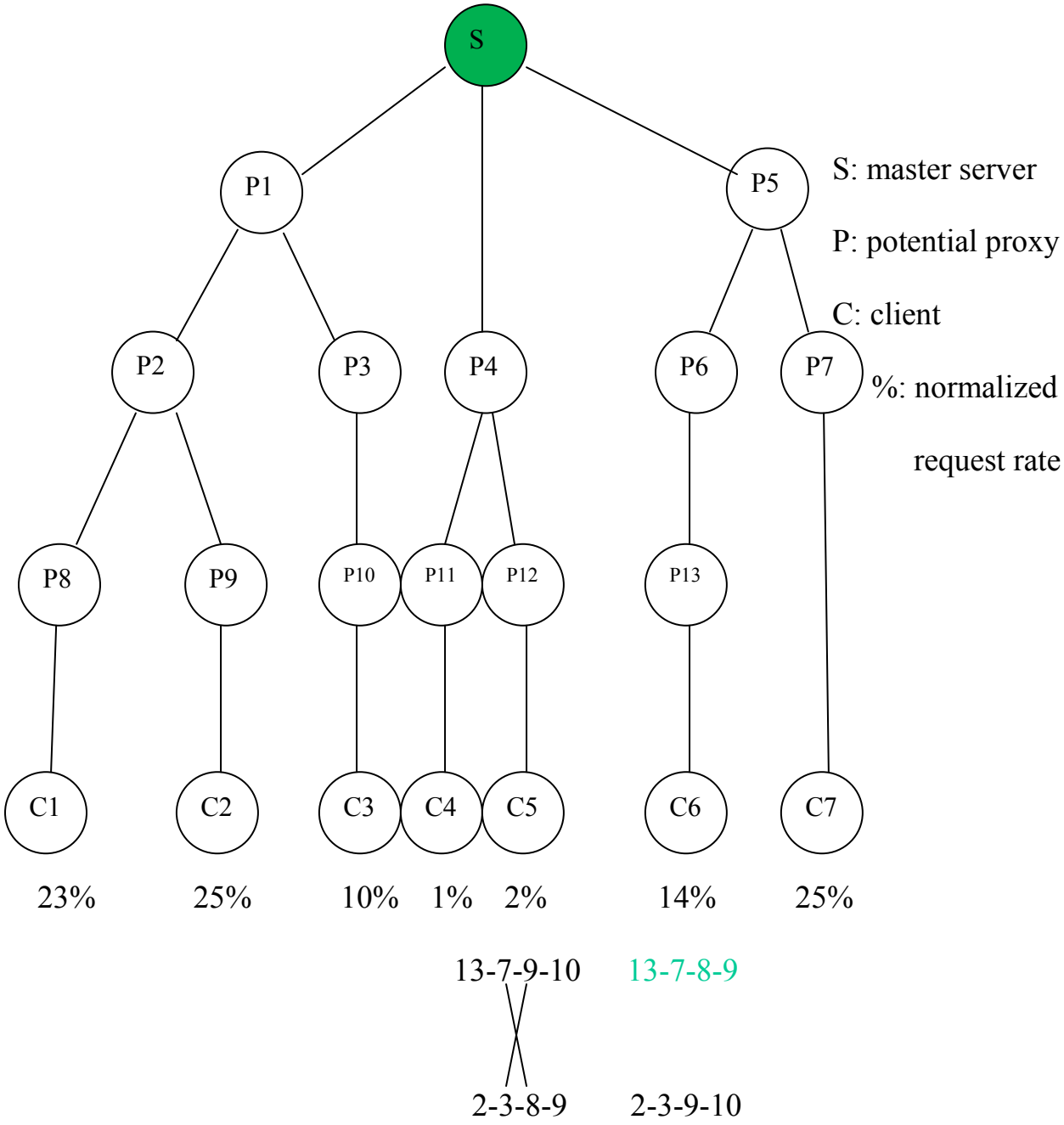


Figure 6: Crossover of Genetic Algorithm

6.5 Similar problems can be solved by genetic algorithm

Genetic algorithm can solve similar optimization problems such as cost optimal multicast routing in data networks, and medical facility placement for health care delivery systems.

Wang et al. [2001] used genetic algorithm to solve the quality of service (QoS) multicast routing problem that are dependent on two factors: (1) bounded end-to-end delay and link bandwidth along the paths from the original source to destination, and (2) minimum cost of the multicast tree, where the link delay and the link cost are independent metrics. Computing such a constrained multicast tree is NP-complete. Their experiments show that their proposed genetic algorithm is efficient and effective.

Haghighat et al. [2004] proposed several solutions to the bandwidth-delay-constrained minimum cost multicast routing problem by using genetic algorithms (GA). They proposed six different coding schemes for chromosome representation, and also some new heuristic algorithms for genetic operator such as mutation, crossover, and initial random populations. They evaluate the performance and efficiency of the proposed algorithms in comparison with other existing heuristic using simulation experiment results.

The most efficient combination of various proposed alternative algorithms is selected as their final solution by the simulation experiment results. The proposed genetic algorithm is better than the existing algorithms in terms of average tree cost and running time.

Lu et al. [2010] stated that a complete supply chain include the forward and reverse logistics. Therefore both original forward logistics system and the reverse logistics are taken into account, the one-way logistics will be transformed into two-way logistics under the premise of the most savings capital, and the logistics center location problem model is established. An improved genetic algorithm is put forward, and makes it solving the model. Because considering a number of environmental factors, the model makes itself more convenient for real world applications.

Ting et al. [2003] developed a mathematical model for the dynamic location problem where future demands are changing. Given the final number of facilities to be opened, we must locate these facilities at optimality such that the objective function is minimized. They assumed that facilities opened in the site decision would not be closed. That assumption is valid for many location problems given the expense and durability of many facilities. Even the most basic facility location models is a NP-hard problem which is difficult to solve. That is why genetic algorithm (GA) is proposed

to analyzing such problems. Two objects, location and opening time (where and when) of those facilities, are optimized simultaneously.

7. Simulation result for a simple topology

In our simulation, each path from client to master sever forms a linked list where each linked list node is a object which has two date members, one integer value of network node label and another Boolean variable represent if the network node is a proxy. After that insert the head pointer of each linked list to a new liked list whose head point called topology handle, through topology handle we can access each path (from client to master server) by traverse each linked list.

Each population of genetic algorithm is an array of size "proxy number +1", the first element is the cost for that particular proxy placement, the rest elements of the array store the labels of the proxy placement. When each population is inserted into a linked list, these populations will be sorted by their cost. For the simple topology and client access pattern in figure 6, we have 13 potential proxy sites, 7 client sites, request rate (normalized) are client 1: 23% , client 2: 25%, client 3 : 10%, client 4: 1%, client 5: 2%, client 6: 14%, client 7: 25%. The simulation results from genetic algorithm

are in table 1. The results from greedy algorithm are in table 2. Compare these two tables the result is encouraging. When proxy number is 1 or 2, the results from the two algorithms are the same. In other cases (proxy numbers are 3, 4, 5) the results from genetic algorithm are out perform results from greedy algorithm (costs are less) . For proxy numbers 1, 2, 3, 4, 5, the results from genetic algorithm happen to be the same as global optimal. The simulation time in each case of genetic algorithm is significantly less than greedy algorithm.

proxy number	initial population	reproduction	cost	simulation time	results
1	60	60	1.76	0.002 sec	2
2	120	30	1.26	0.009 sec	2, 7
3	120	60	0.78	0.012 sec	7, 8, 9
4	60	60	0.36	0.009 sec	7, 8, 9, 13
5	120	30	0.06	0.011 sec	7, 8, 9, 10, 13

Table 1: Results from genetic algorithm

proxy number	cost	simulation time	Result
1	1.76	0.005 sec	2
2	1.26	0.062 sec	2, 7
3	0.84	0.094 sec	2, 7,13
4	0.54	0.141 sec	2, 7, 10, 13
5	0.29	0.172 sec	2, 7, 9, 10, 13

Table 2: Results from greedy algorithm

proxy number	Greedy Algorithm results	Genetic Algorithm Results	Global optimal Results
1	2	2	2
2	2, 7	2, 7	2, 7
3	2, 7,13	7, 8, 9	7, 8, 9
4	2, 7, 10, 13	7, 8, 9, 13	7, 8, 9, 13
5	2, 7, 9, 10, 13	7, 8, 9, 10, 13	7, 8, 9, 10, 13

Table 3: Results from greedy algorithm, genetic algorithm and global optimal solution

We tested a series of different request rate and plot the graph as Figure 7. The x axis is different sets of request rates, the y axis is costs for replica placements. The blue line is for results from greedy algorithm.

Burgundy line is for results from genetic algorithm. In all request rates, greedy algorithm is better than greedy algorithm except for request rate 10. For request rate 10, the costs for replica placement from both algorithms are the same.

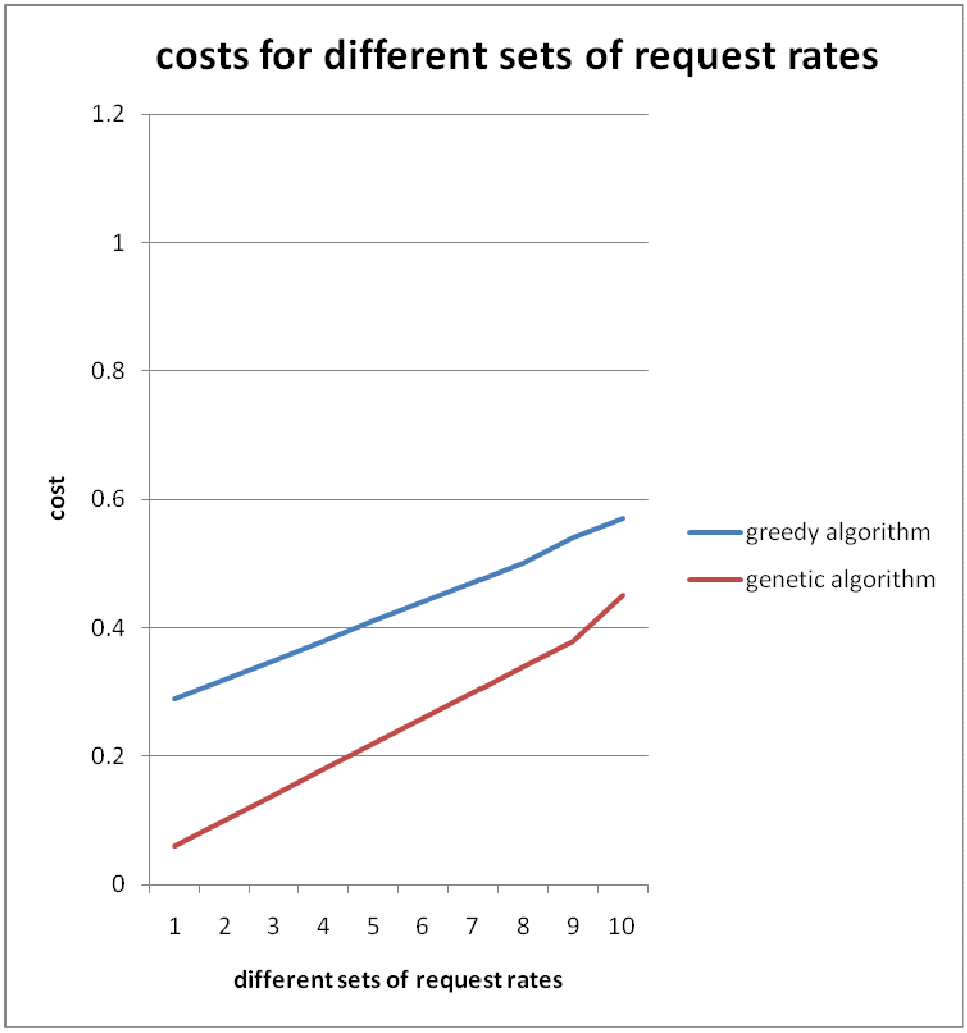


Figure 7: Compare costs between greedy algorithm and genetic algorithm

The user experience (the parameter of cost) is discussed above. The more proxies, the better user experience. But that will contradict with goal of

minimize the resouce consumption. As we already defined the objective function: $F(X) = w*C^{\alpha}(X) + (1-w)* f^{\beta}(X)$ where X is the proxy placement, C(X) is the objective function for user experience, f(X) is the objective function for resource consumption. Here to simplify the problem, we let the $\alpha = \beta = 1$, so the objective function $F(X) = w*C(X) + (1-w)* f(X)$. is a liner combination of cost and resouces consumption. Further we let $l_1 = 0.77$, $l_2 = 0.23$ and use the number of proxies to represent the f(X). Based on this simplification we have the table 4 below:

proxy number	initial population	reproduction	F(X)	results
1	60	60	1.58	2
2	120	30	1.43	2, 7
3	120	60	1.28	7, 8, 9
4	60	60	1.20	7, 8, 9, 13
5	120	30	1.28	7, 8, 9, 10, 13

Table 4: Results from different number of proxies in genetic algorithm

From table 4 we can see the opimal solution for the simpl topology is to choose the proxy number to 4 so that the multi-objective function (combined result of user experience and resouce consumption) is optimal.

8. Simulation results for a large topology

In order to use a realistically large topology for simulation we studied the network map for major U.S. carriers, based on that we designed our model to have 279 nodes, 226 client clusters. Each client has to travel through 4 hops in order to reach the master server. There are 12 national caches, 50 regional caches, 206 institutional caches. The traffic pattern is initially randomly generated and fixed throughout the simulation. In order to calculate the normalized cost, each request rate is normalized. As for the program parameter we set the initial population as 200 and use variable mutation rate and step size. Initially the mutation rate is 0.40 as the cost change decrease we will change mutation rate to 0.20, 0.10 and 0.05 accordingly. We start the step size with 16 and later change to 8, 4, 2 and 1 subsequently.

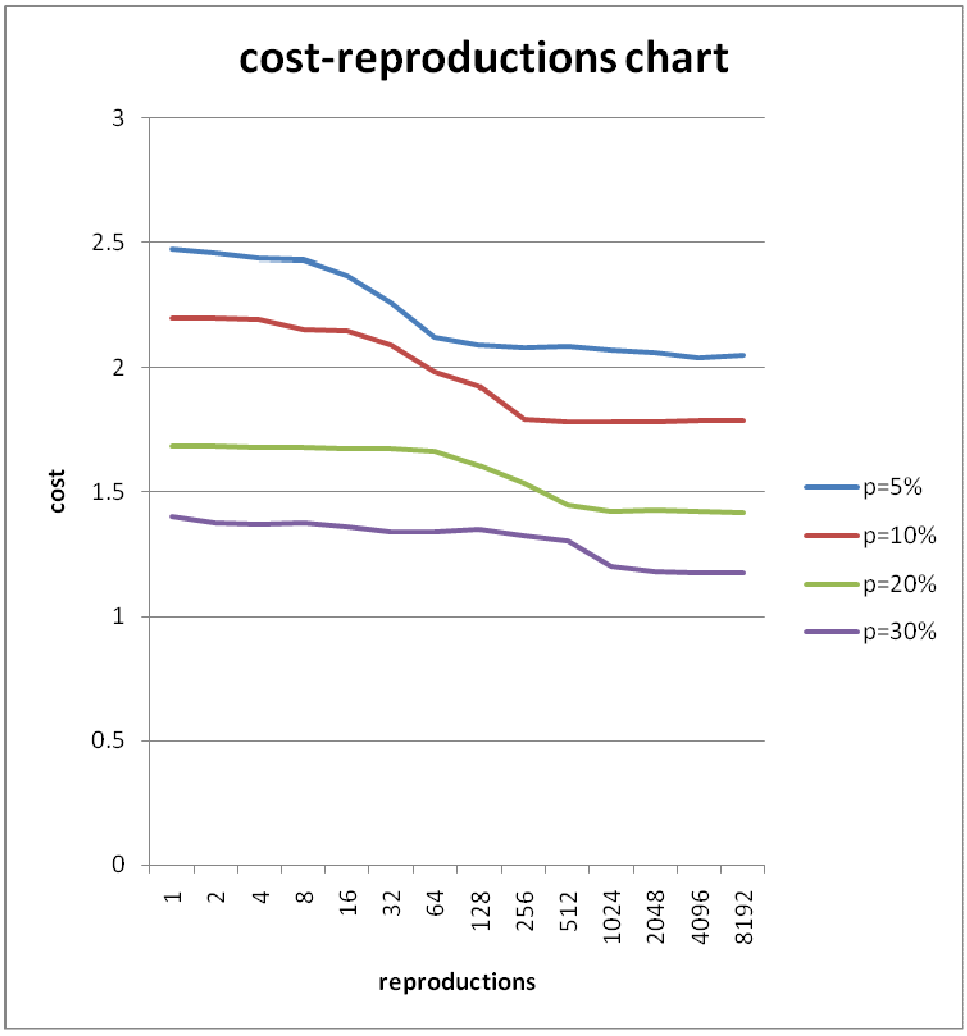


Figure 8: Cost reproductions chart

Figure 7 demonstrate the genetic algorithm behavior. The x-axis of the graph is logarithmic of the number of reproductions, the y-axis is the normalized cost. As the reproduction number increases, the cost for the topologies decreases. The graph represents the results for different percentage values of the proxies (5%, 10%, 20%, 30% of nodes chosen as proxies).

The top line in the graph represents that only 5% of the nodes in the network are chosen as proxies (14 nodes). Compare to other line in the graph, since this case contains fewest proxies, the costs of these topologies is the highest. The genetic algorithm starts to generate significant improvement after 8 reproductions. After 128 reproductions, the genetic algorithm converges towards stable cost.

The second line in the graph represents 10% of the nodes in the network are chosen as proxies (28 nodes). The costs are smaller than the top line. The genetic algorithm starts to generate significant improvement after 16 reproductions. After 256 reproductions, the genetic algorithm converges towards stable cost.

The third line in the graph represents 20% of the nodes in the network are chosen as proxies (56 nodes). The costs are smaller than the second line. The genetic algorithm starts to generate significant improvement after 64 reproductions. After 512 reproductions, the genetic algorithm converges towards stable cost.

The bottom line in the graph represents 30% of the nodes in the network are chosen as proxies (84 nodes). The costs are the smallest. The genetic algorithm starts to generate significant improvement after 256

reproductions. After 512 reproductions, the genetic algorithm converges towards stable cost.

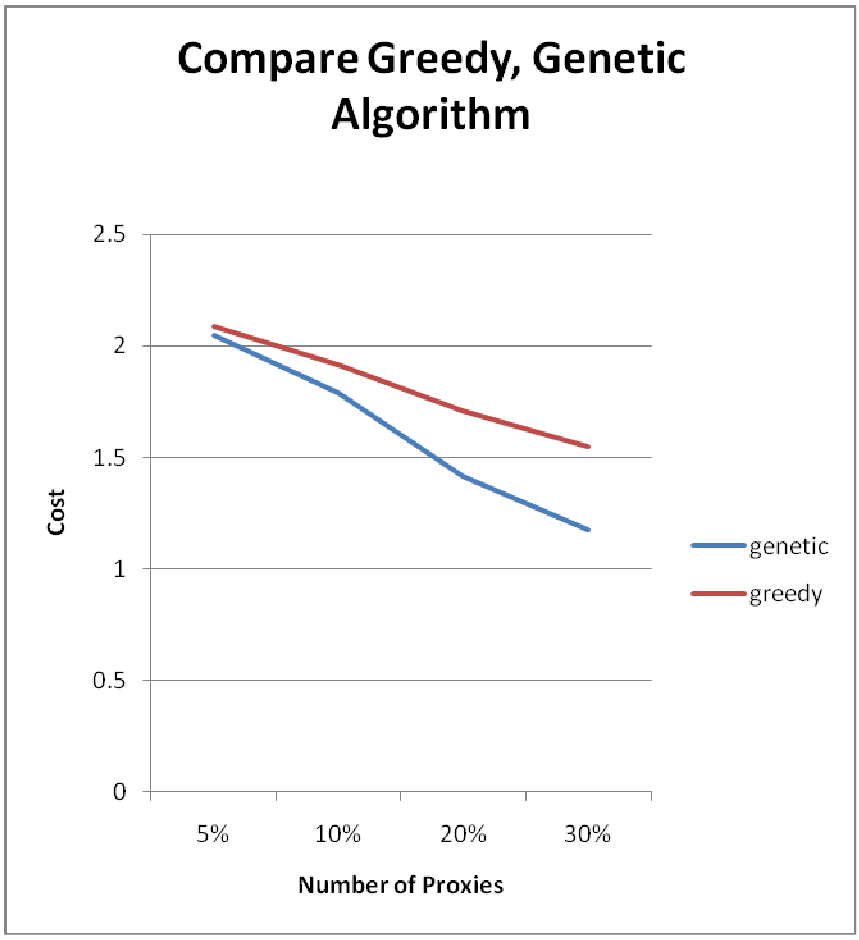


Figure 9. compare genetic algorithm and greedy algorithm

Compare the experiment results between genetic algorithm and greedy algorithm we have the Figure 8. For small proxy numbers, the results from greedy algorithm, genetic algorithm are very close to each other. As the proxy number increase, the genetic algorithm gradually outperform the greedy algorithm. Although the topology is too bigger to use exhaustive

search to get the global optimal solution, at least we know genetic algorithm is better than greedy algorithm, later we will compare genetic algorithm, greedy algorithm with global optimal solution for smaller topologies.

Over all, we have these observations: the more nodes chosen as proxies, the smaller the costs are; the more nodes chosen as proxies, the more reproductions needed for genetic algorithm to generate significant improvement; the more nodes chosen as proxies, the more reproductions needed for genetic algorithm to converge towards stable solution.

When we compare the simulation time between greedy algorithm and genetic algorithm we have Figure 10. Over all, the simulation time for greedy algorithm is much less than genetic algorithm. Especially when we choose more proxy numbers, the difference is even bigger. with my computer (Intel Core Duo processor 2.66GHz, 4GB RAM) the machine time for the simulation is 30.8 minutes while greedy algorithm only needs 4.9 minutes. This is match our theoretical analysis in Chapter 6.

Once again, The more proxies, the better user experience. But that will contradict with goal of minimize the resouce consumption. As we already defined the objective function: $F(X) = l_* C^{\alpha}(X) + (1- l)_* f^{\beta}(X)$ where X is the proxy placement, C(X) is the objective function for user experience,

f(X) is the objective function for resource consumption. Here to simplify the problem, we let the $\alpha = \beta = 1$, so the objective function $F(X) = w*C(X) +(1-w) * f(X)$. is a liner combination of user experience and resouces consumption. Further we let w =0.75 and use the number of proxies to represent the f(X). Based on this simplification we have the graph in Figure 10.

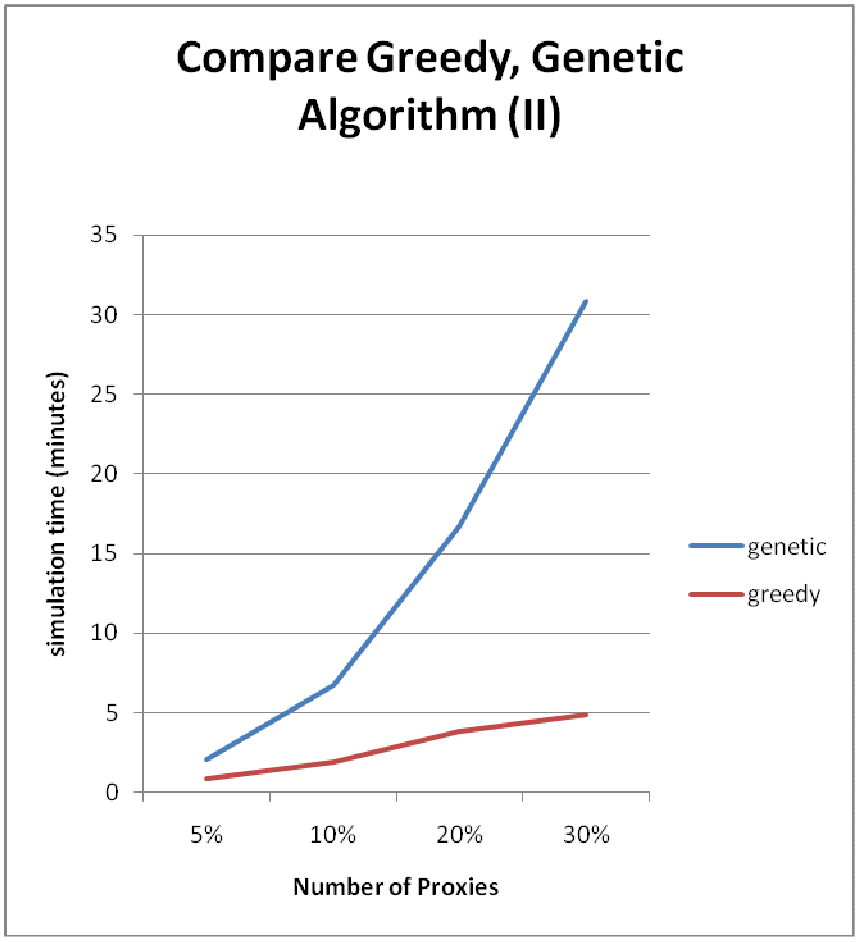


Figure 10. compare genetic algorithm and greedy algorithm(II)

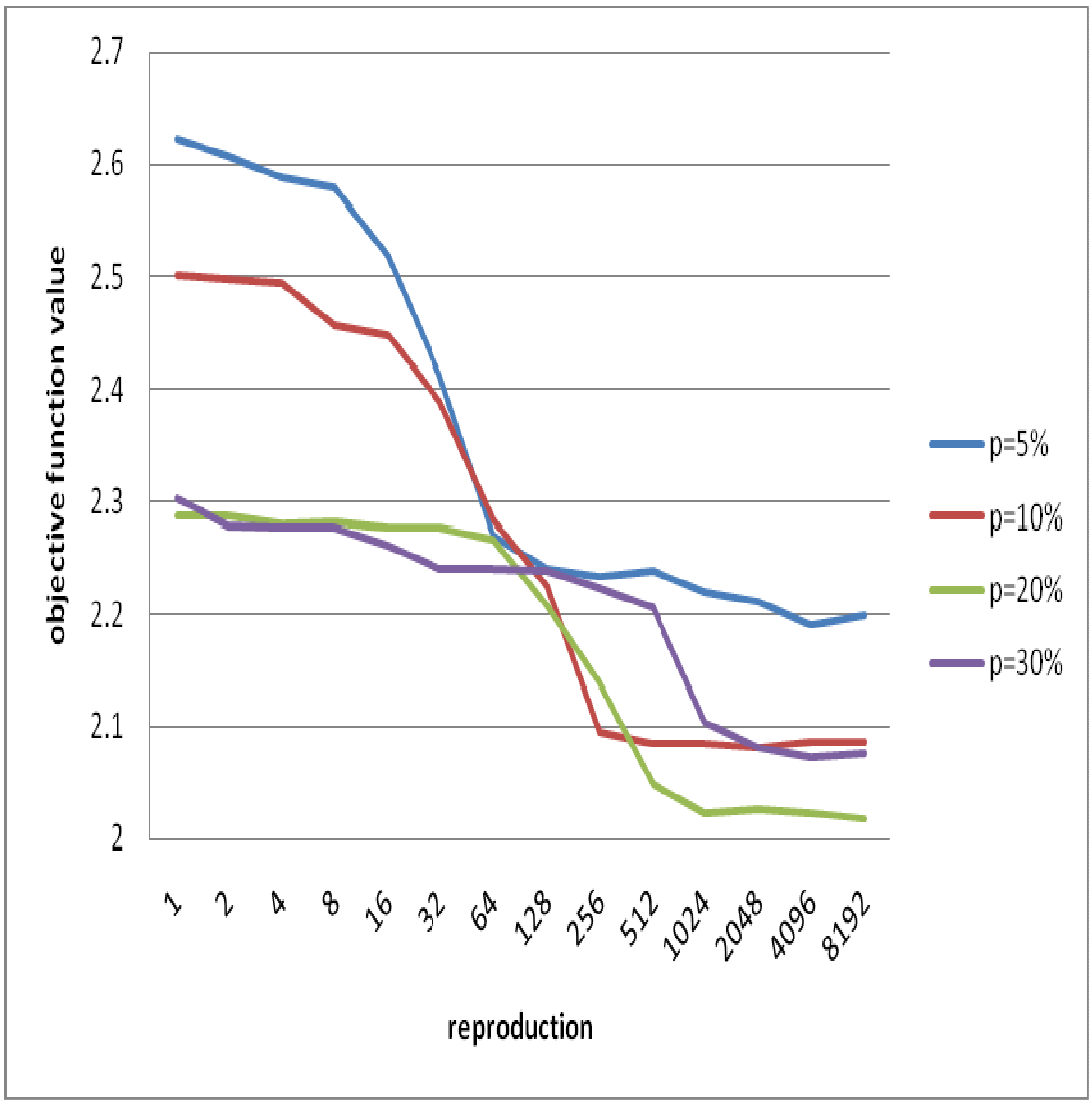


Figure11. Objective function values for a larger topology

From Figure 11 we can see the optimal value for objective function happens when 20% of the nodes are proxies. Although the user experience (latency) is better in the case of 30% of nodes are proxies, it consumes more resources, the overall objective function value is not optimal.

9. Optimizing Genetic Algorithm

In genetic algorithm, we do not have “correct” answer for the genetic parameters. To study the genetic algorithm in the context of proxy placement more broadly, we varied genetic parameters in order to optimize the genetic algorithm. Particularly, we consider the mutation step size, mutation rate, and initial populations.

9.1. Mutation rate

In biology, mutation is the process of randomly changing the genetic substance 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 genetic operation, 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, determines the numerical amount by which a selected proxy site label can change.

9.1.1. Fixed mutation rate

Figure 12 shows the behavior of the genetic algorithm under different mutation rates (all other program parameters remaining constant). First we will experiment with fixed mutation rate. That means the mutation rate will remain the same throughout the entire genetic operation. The purple line represent the mutation rate equals 5%, the burgundy line represent the mutation rate equals 10%, the green line represent the mutation rate equals 20%, the blue line represent the mutation rate equals 30%. No doubt, all the line decrease as the reproductions increase. But we should observe the rate of decrease. Initially, the bigger the mutation rate, the smaller cost. That means higher mutation rates appear to speed up the process of finding better solutions. However, after certain point, the lines turn opposite. The bigger the mutation rate, the higher cost is. That means, higher mutation rate perturbs the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies. The mutation rate of 30% seems to perform best initially, but in the long run, beyond 256 reproductions, the genetic algorithm behaves best with a smallest mutation rate of 5%. One special case: when mutation rate is 0%, that is there is no mutation, the result is worse than all other cases. So mutation is important for genetic algorithm to solve our problem.

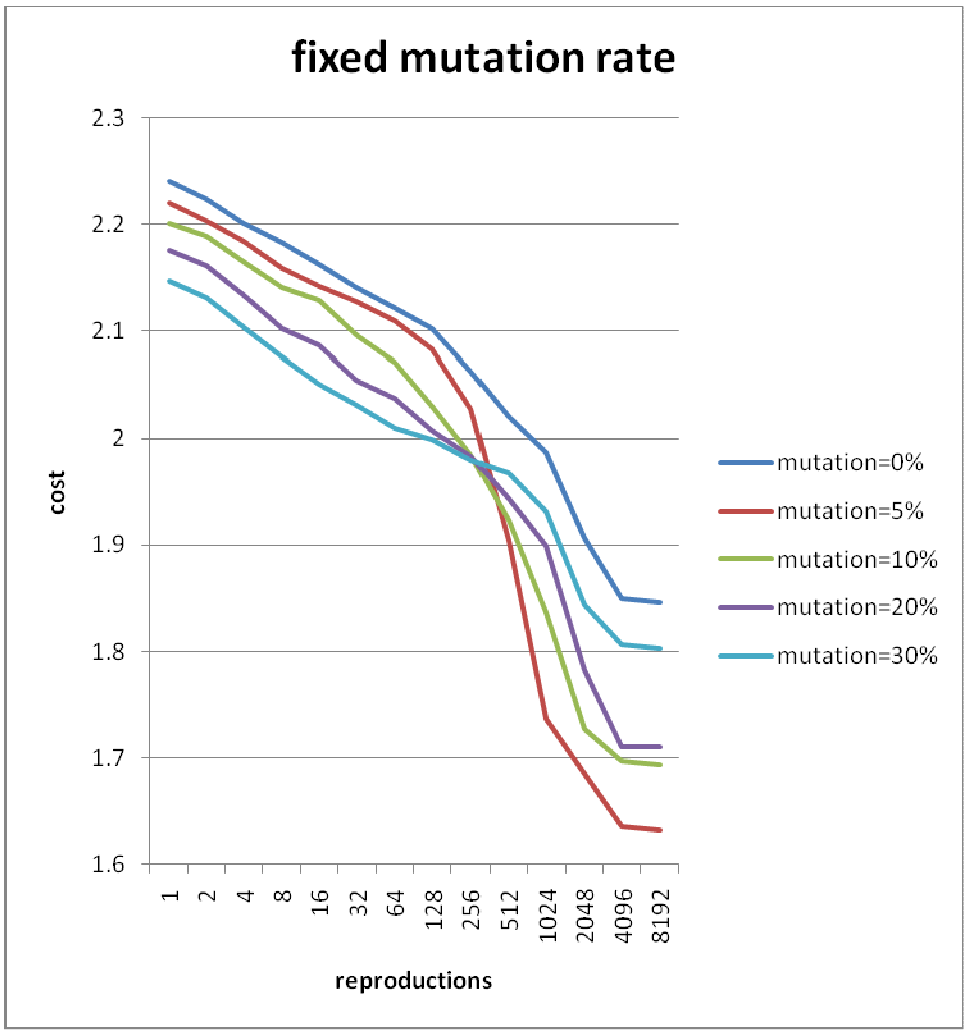


Figure12. Cost reproduction charts under fixed mutation rate

9.1.2. Variable mutation rate

Now we change the mutation rate as we go through the genetic operation. Figure 13~ Figure 16 shows the behavior of the genetic algorithm under different mutation rates (all other program parameters remaining constant). The mutation rate will be cut in half when the genetic operations

go through half reproductions, mutation rate decrease to one fourth when genetic operations go three fourth.

In Figure 12~ Figure 14 we can see the results from variable mutation rates outperform the results from fixed mutation rates. That is because after certain point, higher mutation rates perturb the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies, we need smaller mutation rates to fine tune the genetic operations. Figure 15 is the exception. The results from variable mutation rates are almost the same as the results from fixed mutation rates. That means the 5% mutation rate is small enough that in the final stage of the genetic operations, it will not perturb the genetic operation from quickly converge to the optimal solutions, it doesn't need the variable mutation rate scheme to fine tune the genetic operation.

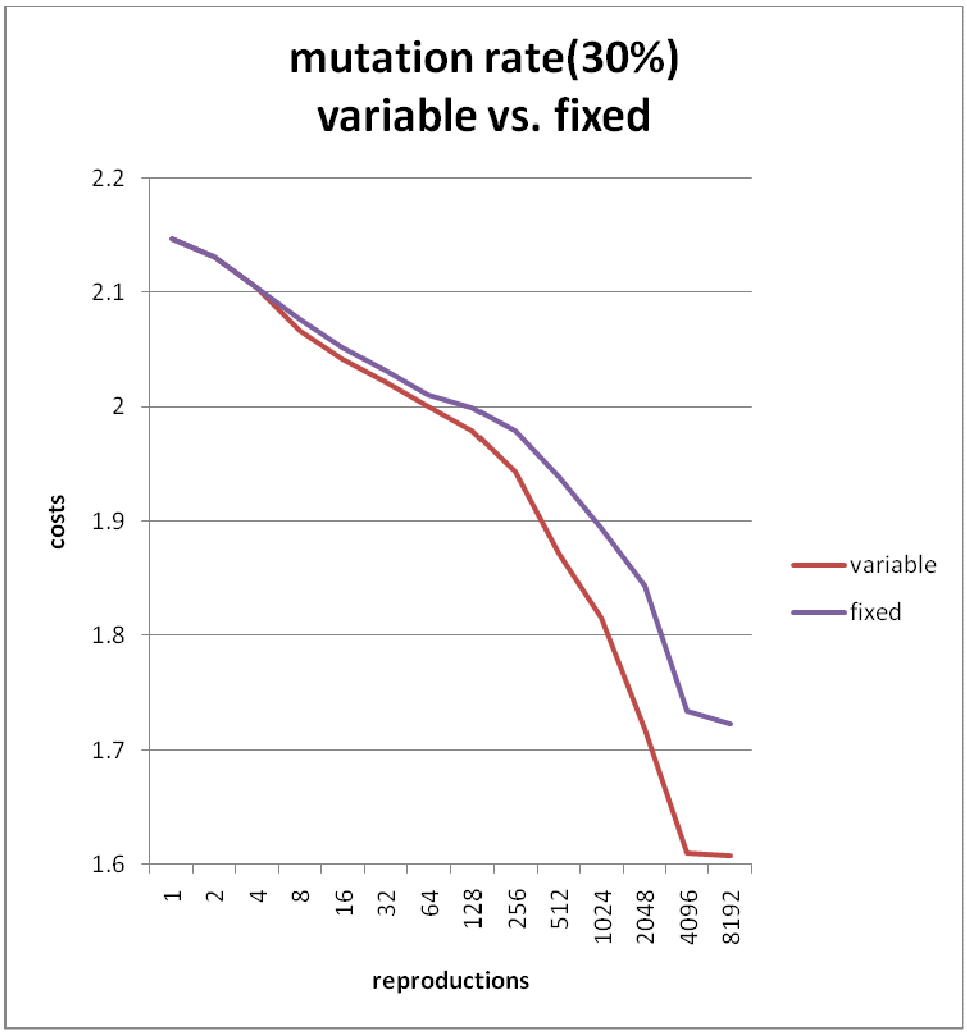


Figure 13: Mutation rate (30%) variable vs. fixed

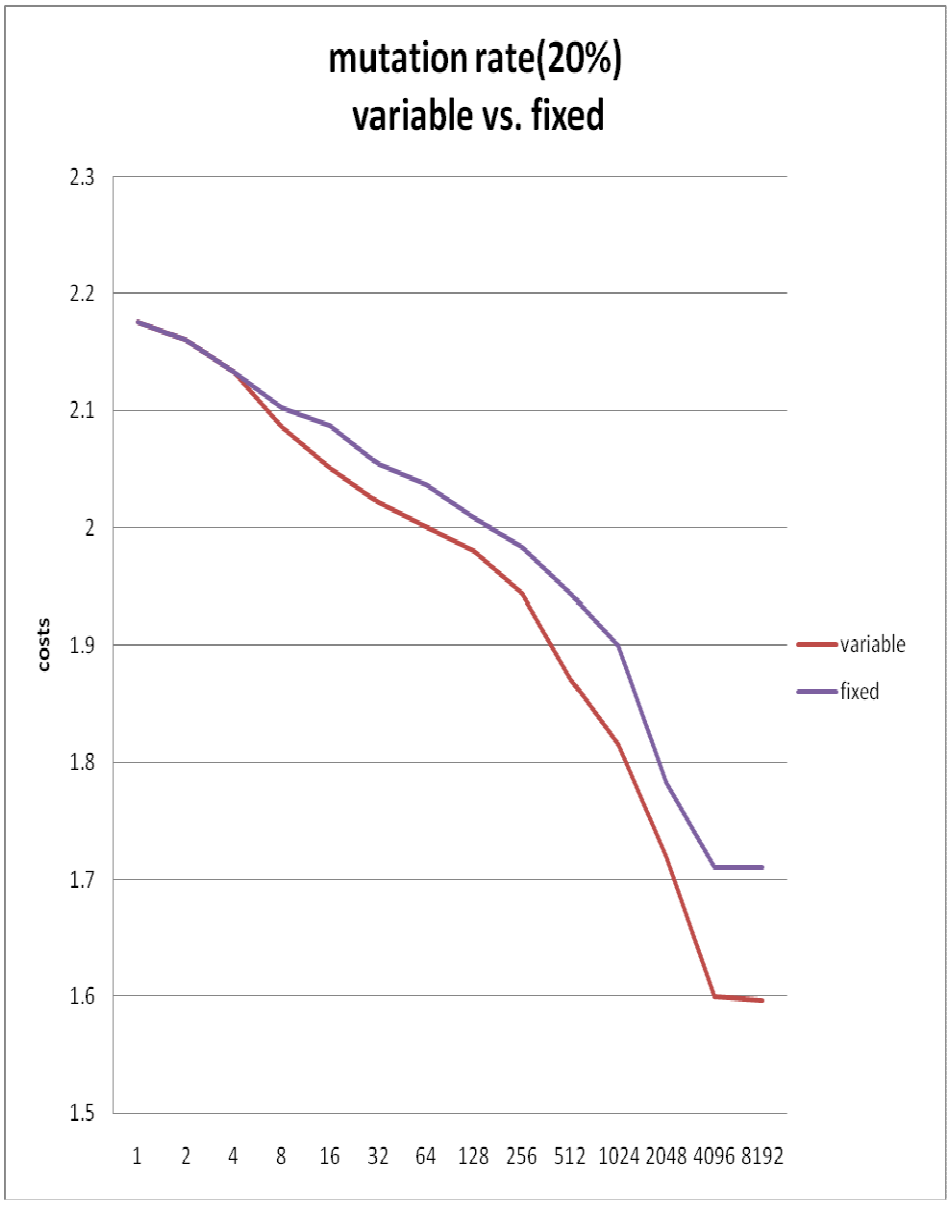


Figure 14: Mutation rate (20%) variable rate vs. fixed rate

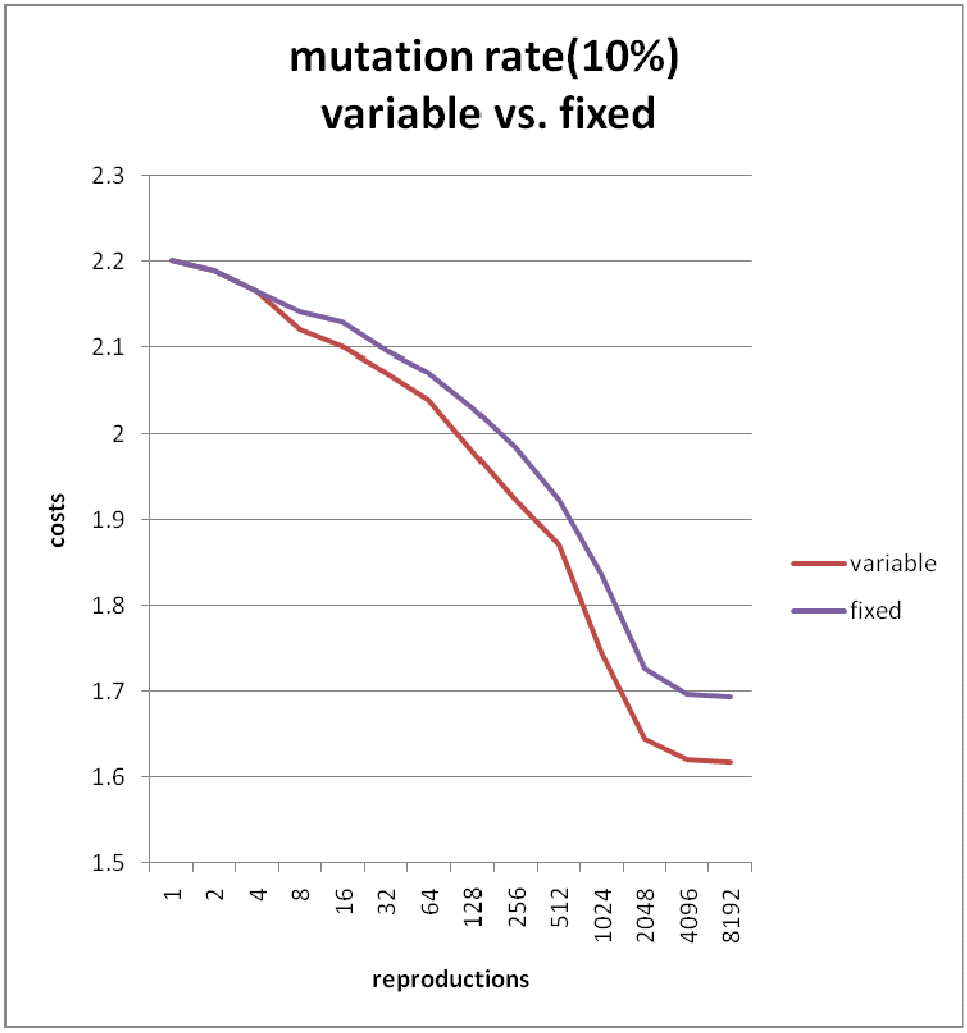


Figure 15: Mutation rate (10%) variable vs. fixed

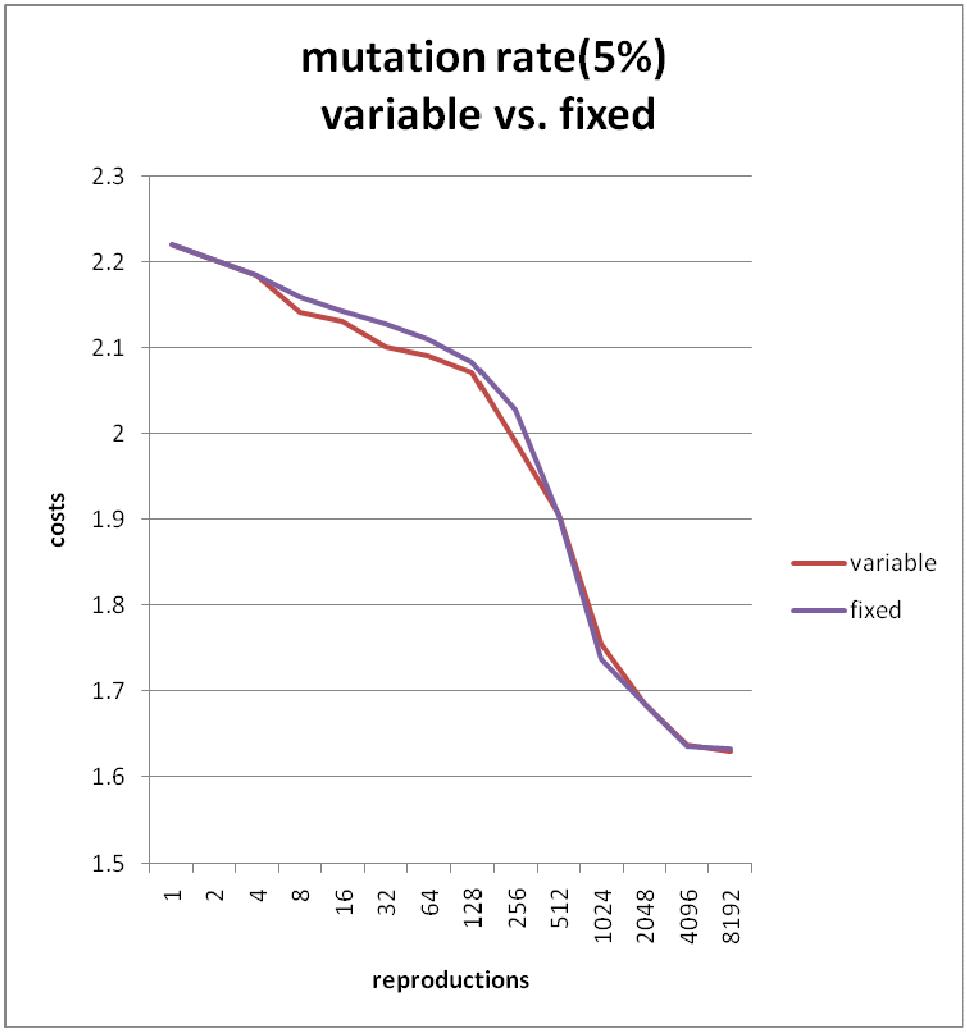


Figure 16. Mutation rate (5%) variable vs. fixed

When we put all the results from four different variable mutation rates, we found at the beginning the bigger the mutation rate the smaller the cost, this is because bigger initial mutation rate can speed up towards the optimal solution. After certain reproductions, there are no big differences for

the results. The variable mutation rate scheme works better than the fixed mutation rate scheme especially when we do not want to wait too long to converge to the results when the access pattern of a large topology suddenly changes. See Figure 17.

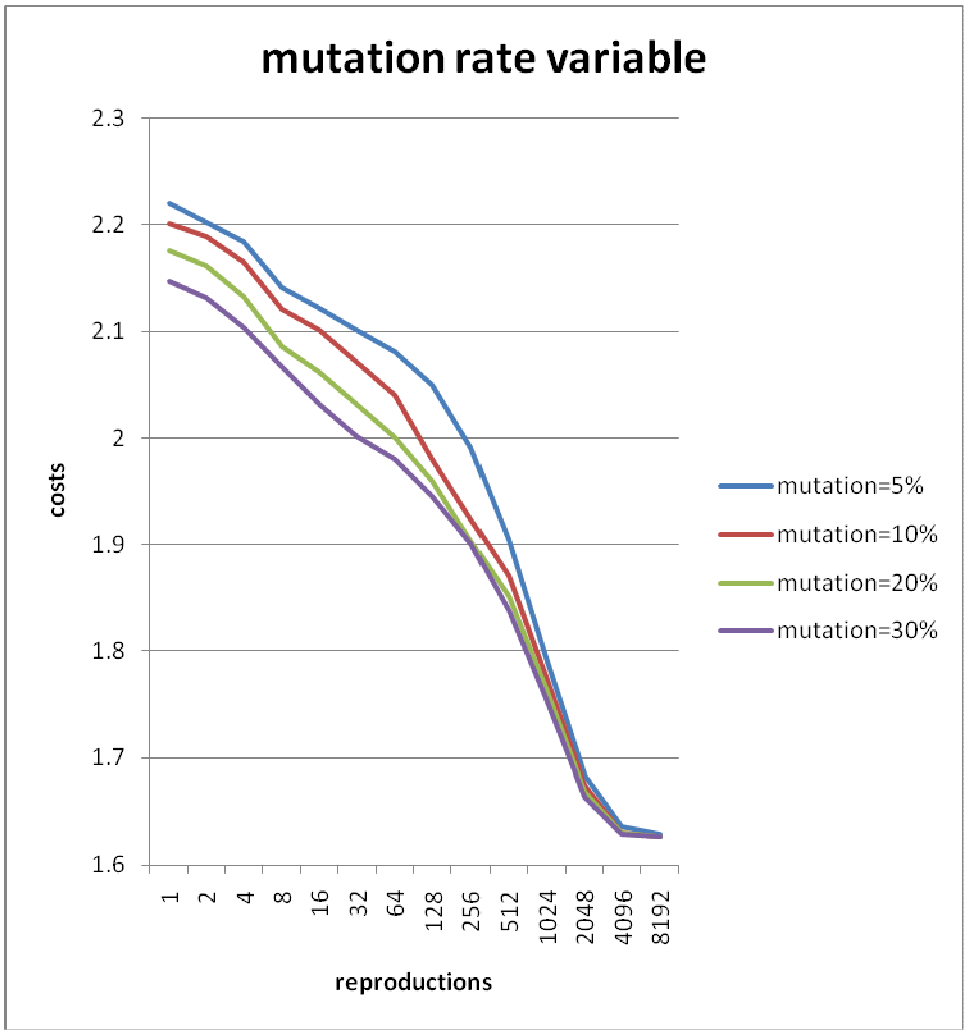


Figure 17: Variable mutation rates

9.2. Mutation step size

9.2.1 Fixed mutation step size

The step size, determines the numerical amount by which a selected proxy site label can change. Figure 18 shows the behavior of the genetic algorithm under different step sizes. First we will experiment with fixed mutation rate. That means the step size will remain the same throughout the entire genetic operation.

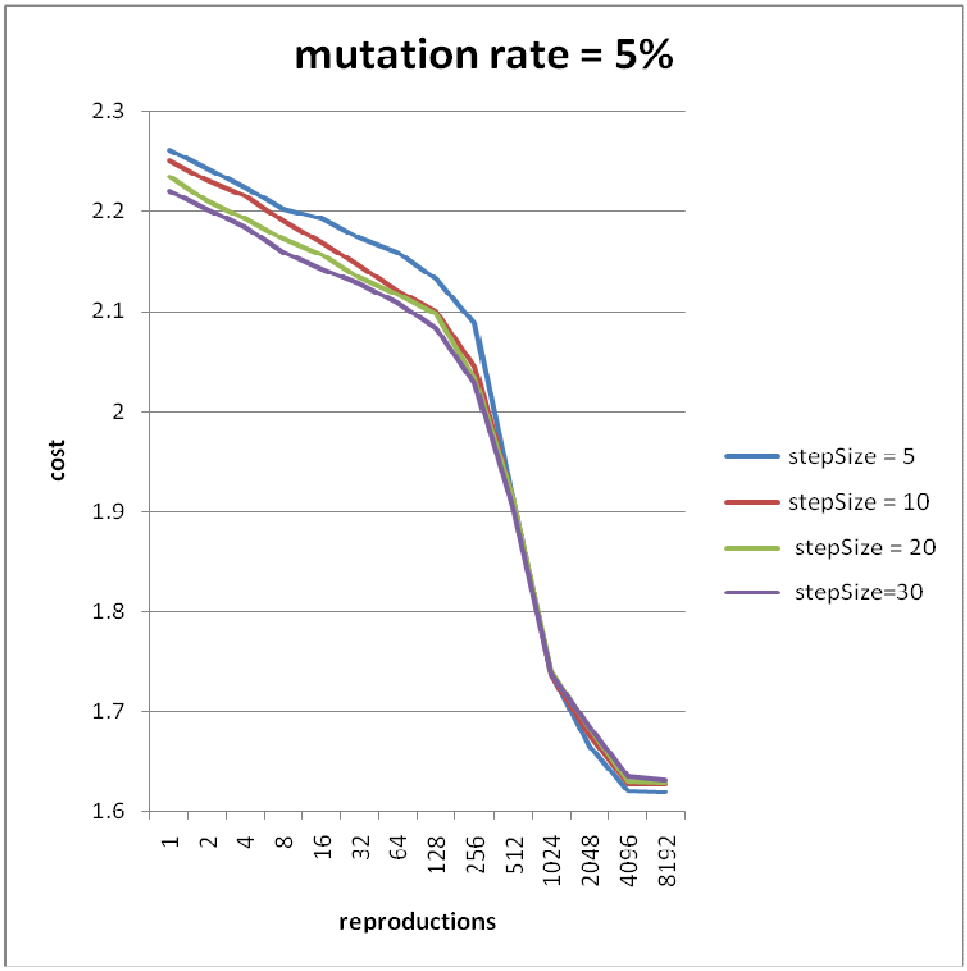


Figure 18. Genetic algorithm sensitivity to step size (mutation rate = 5%)

Figure 18 is the graph show the genetic algorithm sensitivity to step size where mutation rate is fixed to 5%. The purple line represent the step size equals 5, the burgundy line represent the step size equals 10, the green line represent the step size equals 20, the blue line represent the step size equals 30. No doubt, all the line decrease as the reproductions increase. But we should observe the rate of decrease. Initially, the bigger the step size, the smaller cost. That means bigger step size appear to speed up the process of finding better solutions. However, after certain point, the lines turn opposite. The bigger the step size, the higher cost it is. That means, higher step size perturbs the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies. The step size of 30 seems to perform best initially, but in the long run, beyond 512 reproductions, the genetic algorithm behaves best with a smallest step size of 5. However, the difference of the costs is not very big because the mutation rate is quite low.

Similar to Figure 18, Figure 19 is the graph show the genetic algorithm sensitivity to step size where mutation rate is fixed to 10%. The purple line represent the step size equals 5, the burgundy line represent the step size equals 10, the green line represent the step size equals 20, the blue line represent the step size equals 30. No doubt, all the line decrease as the reproductions increase. But we should observe the rate of decrease. Initially,

the bigger the step size, the smaller cost. That means bigger step size appear to speed up the process of finding better solutions. However, after certain point, the lines turn opposite. The bigger the step size, the higher cost it is. That means, higher step size perturbs the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies. The step size of 30 seems to perform best initially, but in the long run, beyond 1024 reproductions, the genetic algorithm behaves best with a smallest step size of 5. In this case, the difference of costs is bigger because the mutation rate is bigger than previous case.

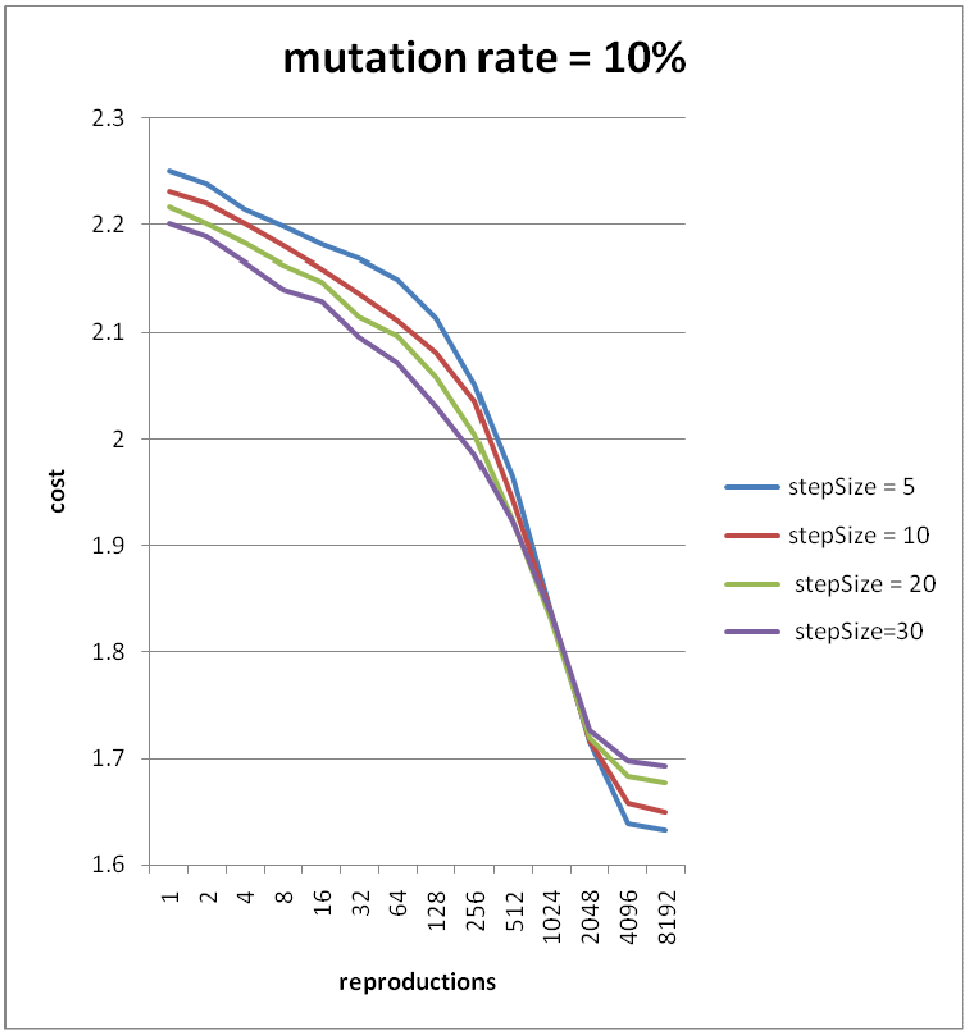


Figure 19: Genetic algorithm sensitivity to step size (mutation rate = 10%)

Simmilar to previous two graphs, Figure 19 is the graph show the genetic algorithm sensitivity to step size where mutation rate is fixed to 10%. The purple line represent the step size equals 5, the burgundy line represent the step size equals 10, the green line represent the step size equals 20, the blue line represent the step size equals 30. No doubt, all the line decrease as the reproductions increase. But we should observe the rate of decrease. Initially, the bigger the step size, the smaller cost. That means bigger step size appear to speed up the process of finding better solutions.

However, after certain point, the lines turn opposite. The bigger the step size, the higher cost it is. That means, higher step size perturbs the genetic algorithm, preventing the population from quickly converge to the

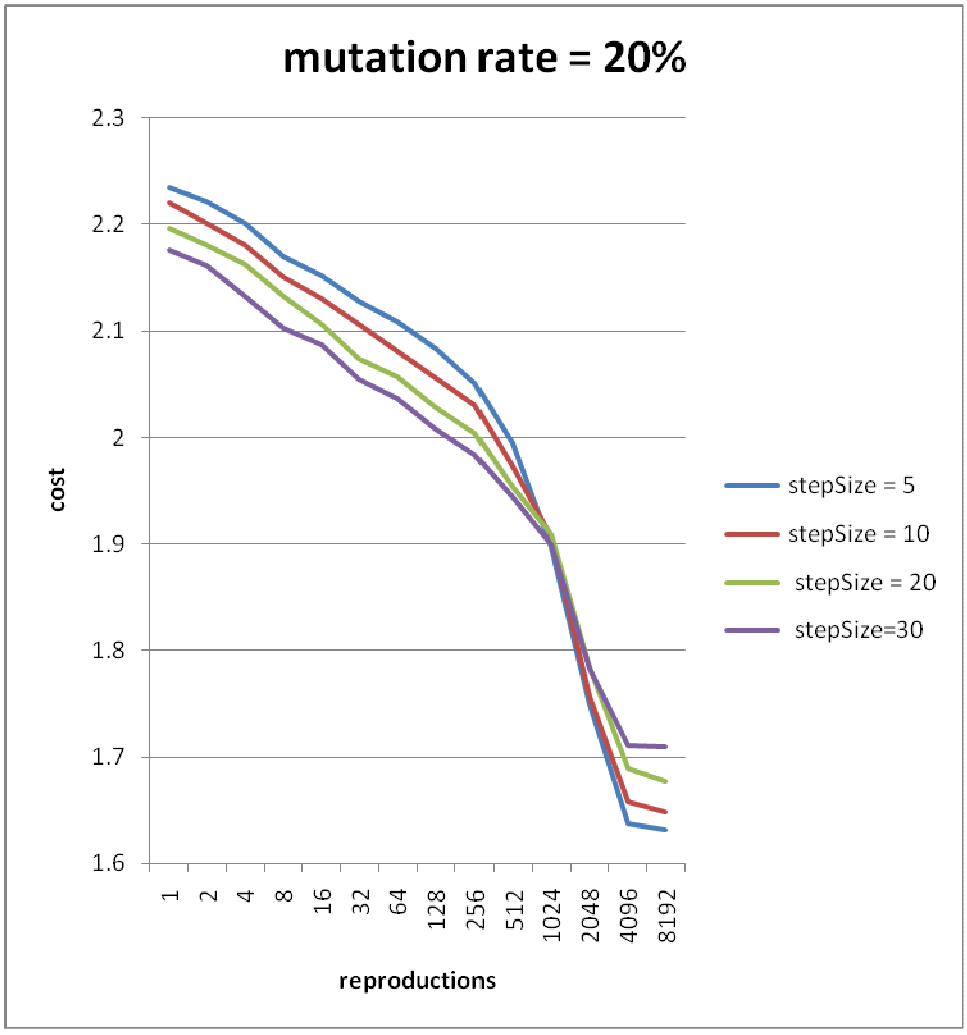


Figure 20. Genetic algorithm sensitivity to step size (mutation rate = 20%)

the long run, beyond 1024 reproductions, the genetic algorithm behaves best with a smallest step size of 5. In this case, the difference of costs is bigger because the mutation rate is bigger than previous cases.

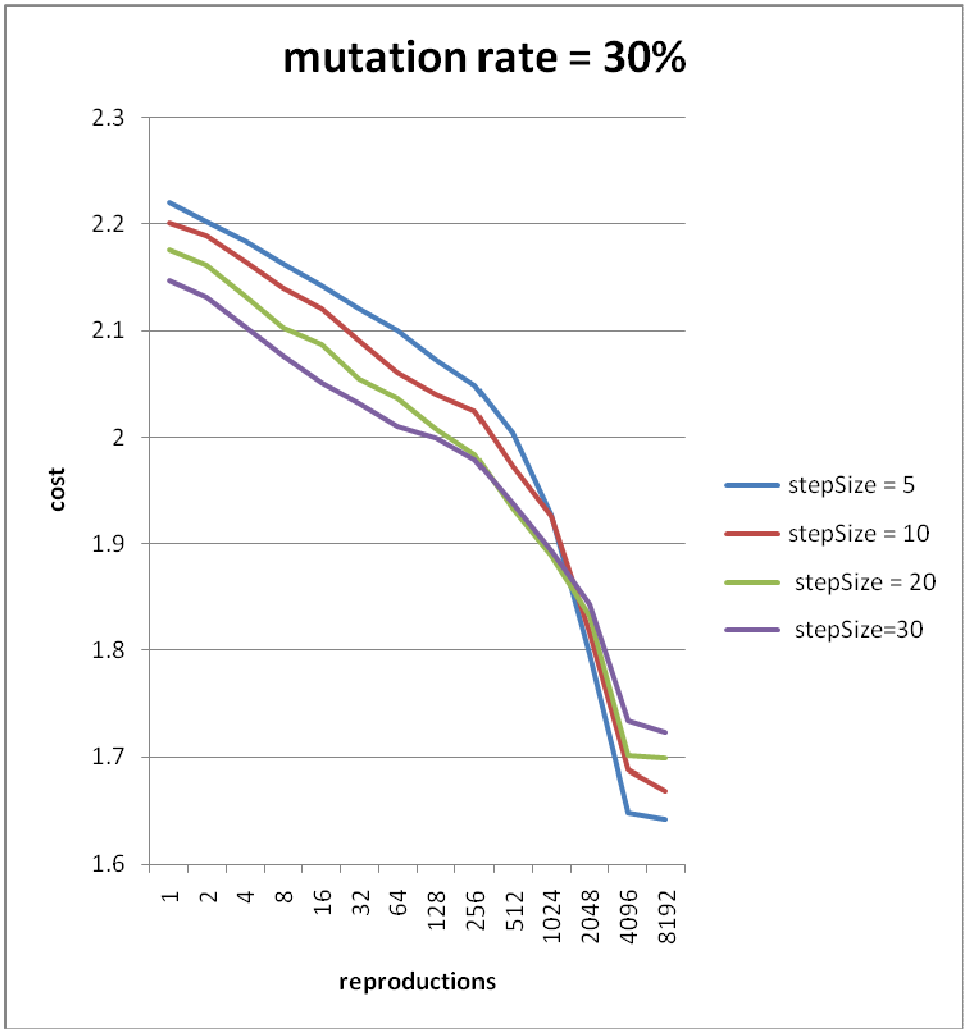


Figure 21. Genetic algorithm sensitivity to step size (mutation rate = 30%)

Finally, Figure 20, Figure 21 is the graph show the genetic algorithm sensitivity to step size where mutation rate is fixed to 20% and 30%. The purple line represent the step size equals 5, the burgundy line represent the

step size equals 10, the green line represent the step size equals 20, the blue line represent the step size equals 30. No doubt, all the line decrease as the reproductions increase. But we should observe the rate of decrease. Initially, the bigger the step size, the smaller cost. That means bigger step size appear to speed up the process of finding better solutions. However, after certain point, the lines turn opposite. The bigger the step size, the higher cost it is. That means, higher step size perturbs the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies. The step size of 30 seems to perform best initially, but in the long run, beyond 1024 reproductions, the genetic algorithm behaves best with a smallest step size of 5. In Figure 21, the difference of costs is the biggest because the mutation rate (30%) is the biggest.

9.2.2 Mutation rate under different step size

Now let's compare the mutation rate sensitivity under different step size. Figure 22 ~ Figure 25 show the genetic algorithm behavior under different step size. The features of all the graphs are the same. The costs of the topologies from genetic algorithms with bigger mutation rates are smaller at the beginning, that's understandable, bigger mutation rates speed

up the process to find the better solutions. Compare Figure 22 ~ Figure 25, we can see the difference between the costs under different mutation rates increase as the step size increases from 5 to 30. After certain number of reproduction, the results turn opposite. The costs of the topologies from genetic algorithms with bigger mutation rates are bigger. When most of the optimal proxy locations are found, bigger mutation rate will perturb the algorithm, after that point small mutation rate can fine tune the process, lead to converge to the optimal proxy location set. Once again, compare Figure 22 ~ Figure 25, we can see the differences between the costs under different mutation rates increase as the step size increases from 5 to 30.

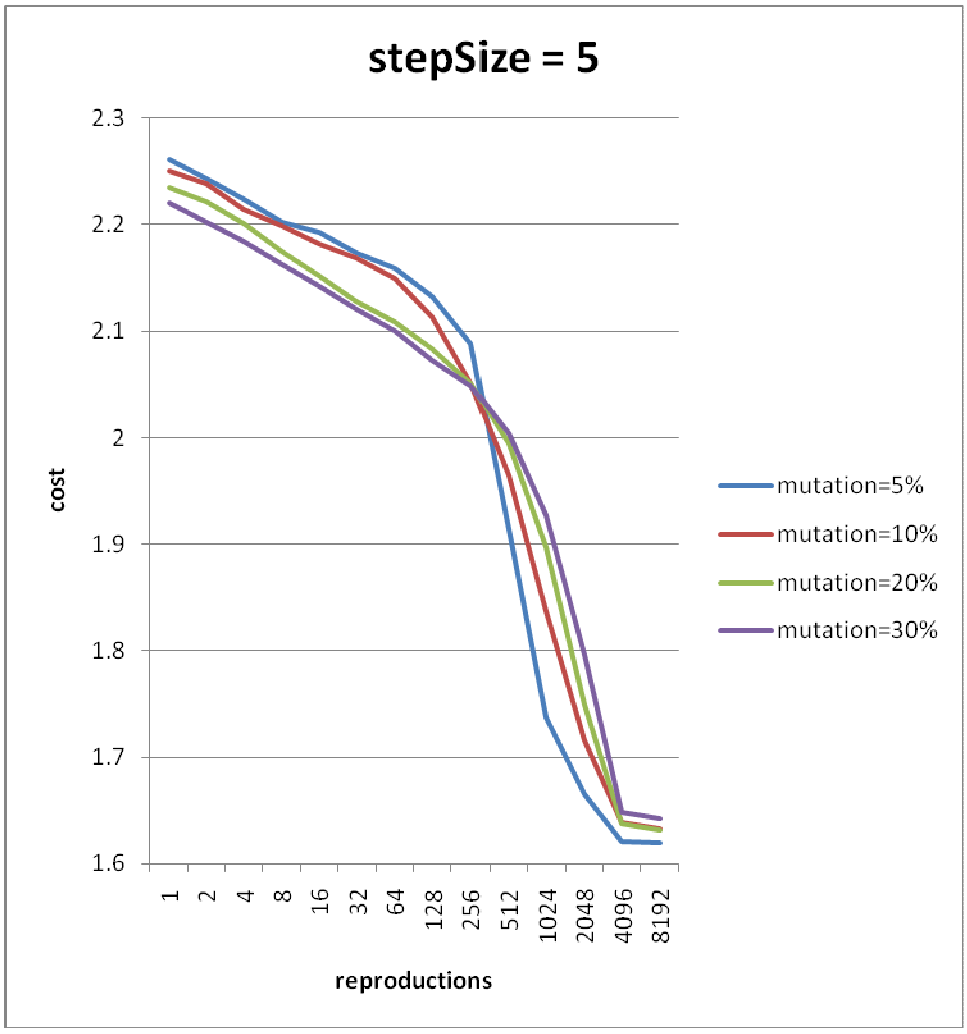


Figure 22. Genetic algorithm sensitivity to mutation rate (step size = 5)

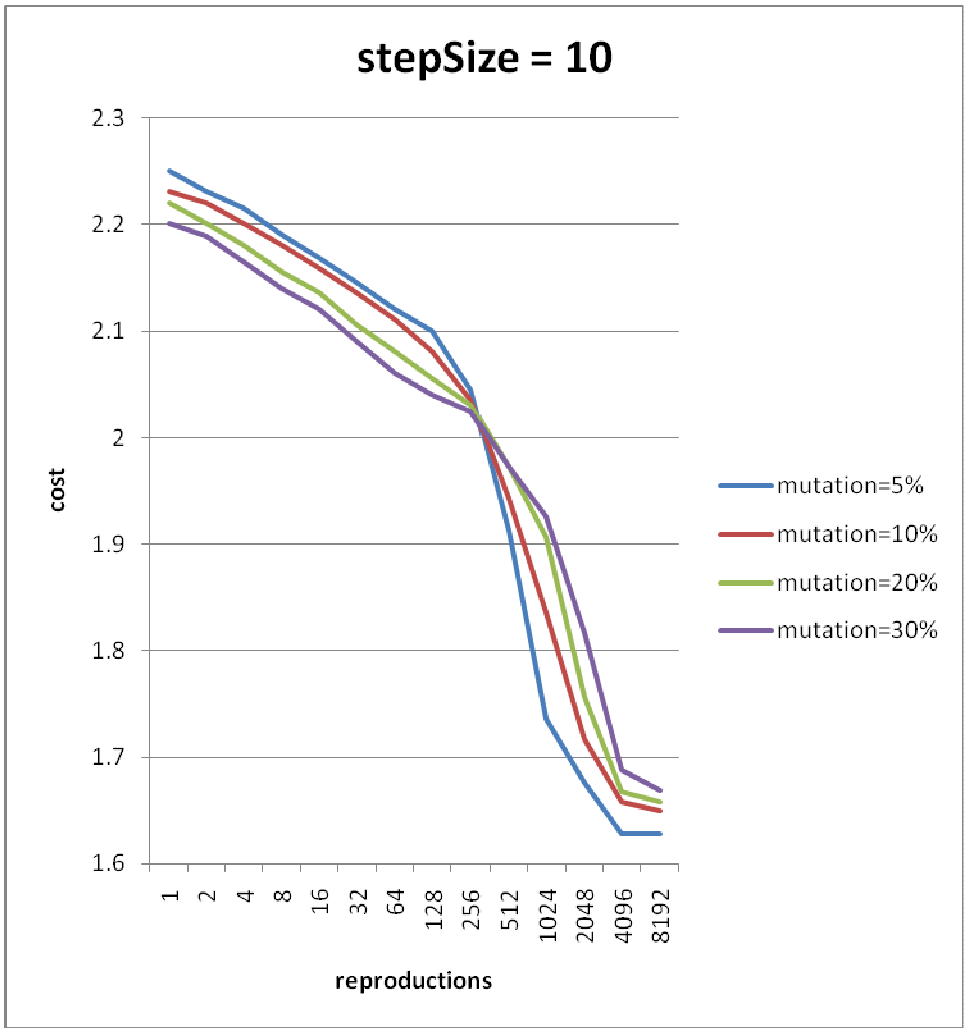


Figure 23. Genetic algorithm sensitivity to mutation rate (step size = 10)

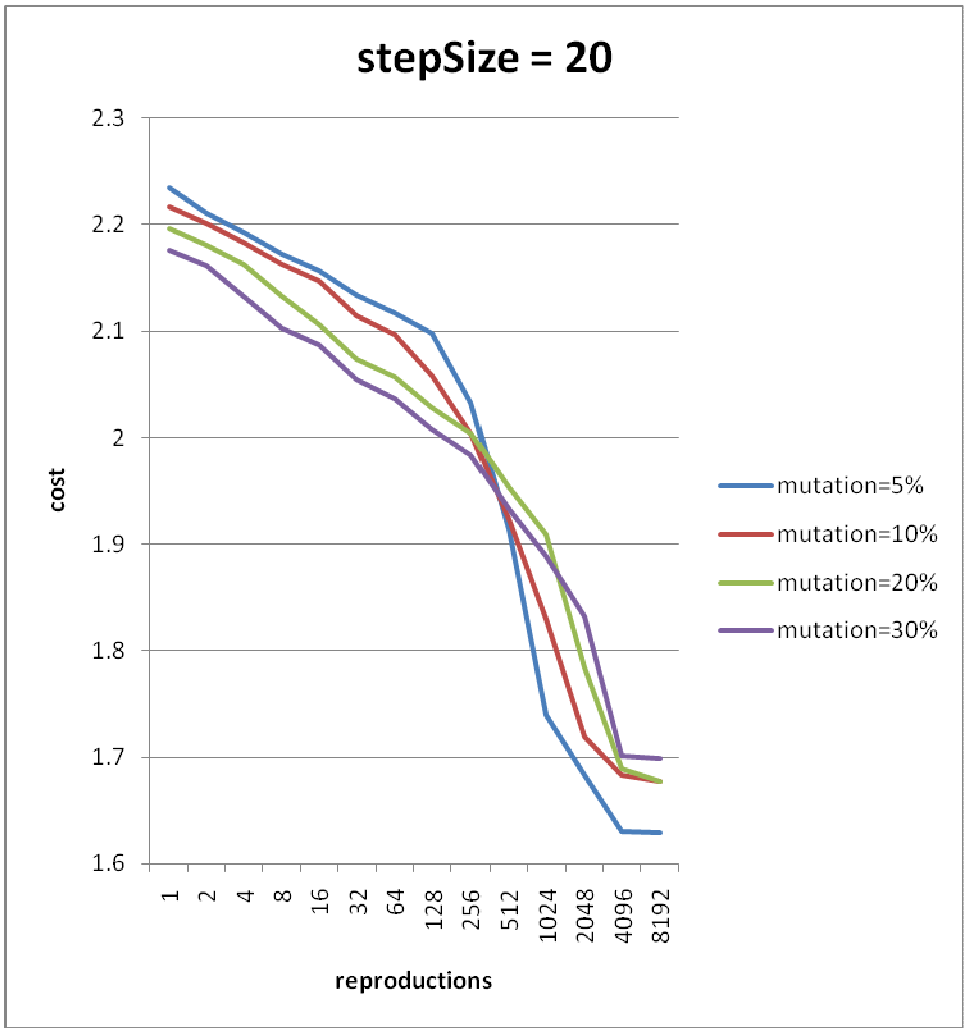


Figure 24. Genetic algorithm sensitivity to mutation rate (step size = 20)

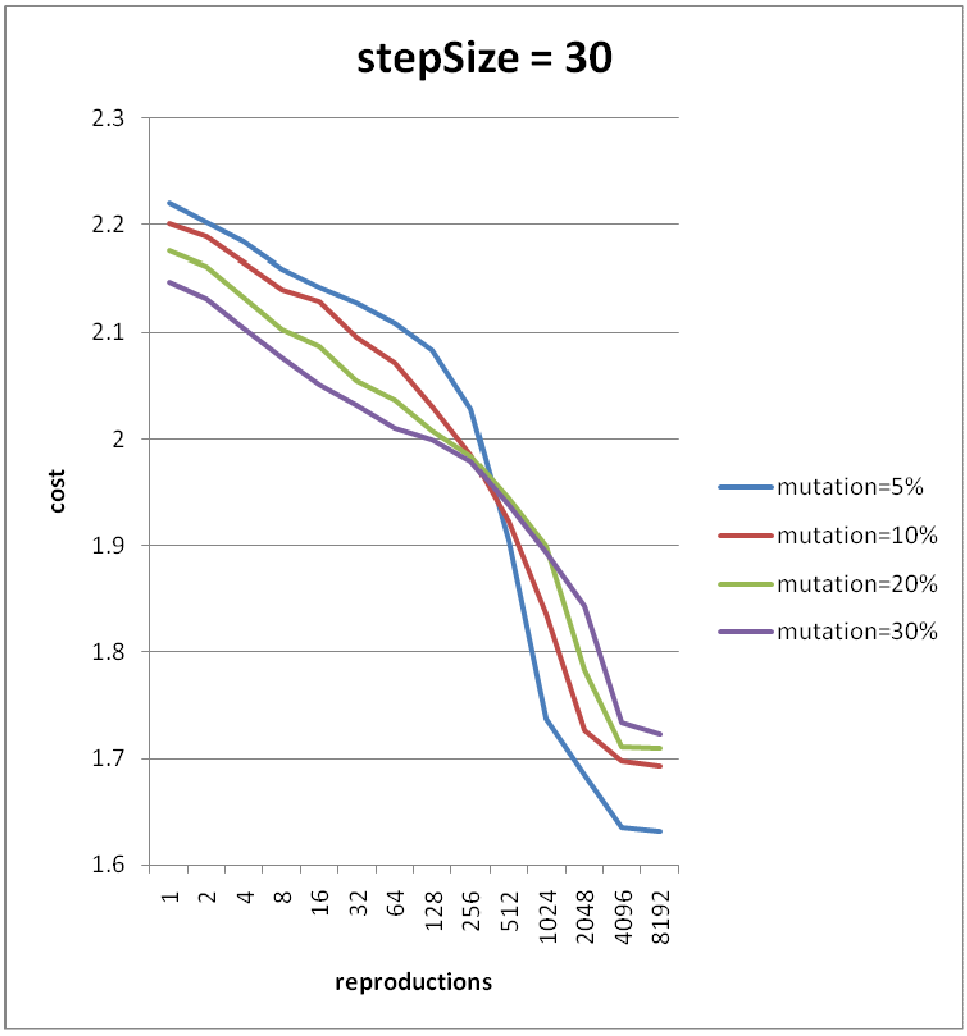


Figure 25. Genetic algorithm sensitivity to mutation rate (step size = 30)

9.3 Variable mutation step size

In previous experiments the step sizes are fixed, now we change the step size as we go through the genetic operation. Figure 26~Figure 29 shows the behavior of the genetic algorithm under different step sizes (all other program parameters remaining constant). The step sizes will be cut in

half when the genetic operations go through half reproductions, step sizes decrease to one fourth when genetic operations go three fourth. The mutation rates in all these experiments are 30%.

The results are very similar to variable mutation rate. In Figure 26~ Figure 28 we can see the results from variable step sizes outperform the results from fixed step size. That is because after certain point, higher step size perturb the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies, we need smaller step size to fine tune the genetic operations, that means after major proxy locations have been found, we need to fine tune to find nearby locations so that the solution will converge to the optimal set of proxies. Figure 29 is the exception. The results from variable step size are almost the same as the results from fixed step size. That means the 5% step size is small enough that in the final stage of the genetic operations, it will not perturb the genetic operation from quickly converge to the optimal solutions, it doesn't need the variable step size scheme to fine tune the genetic operation.

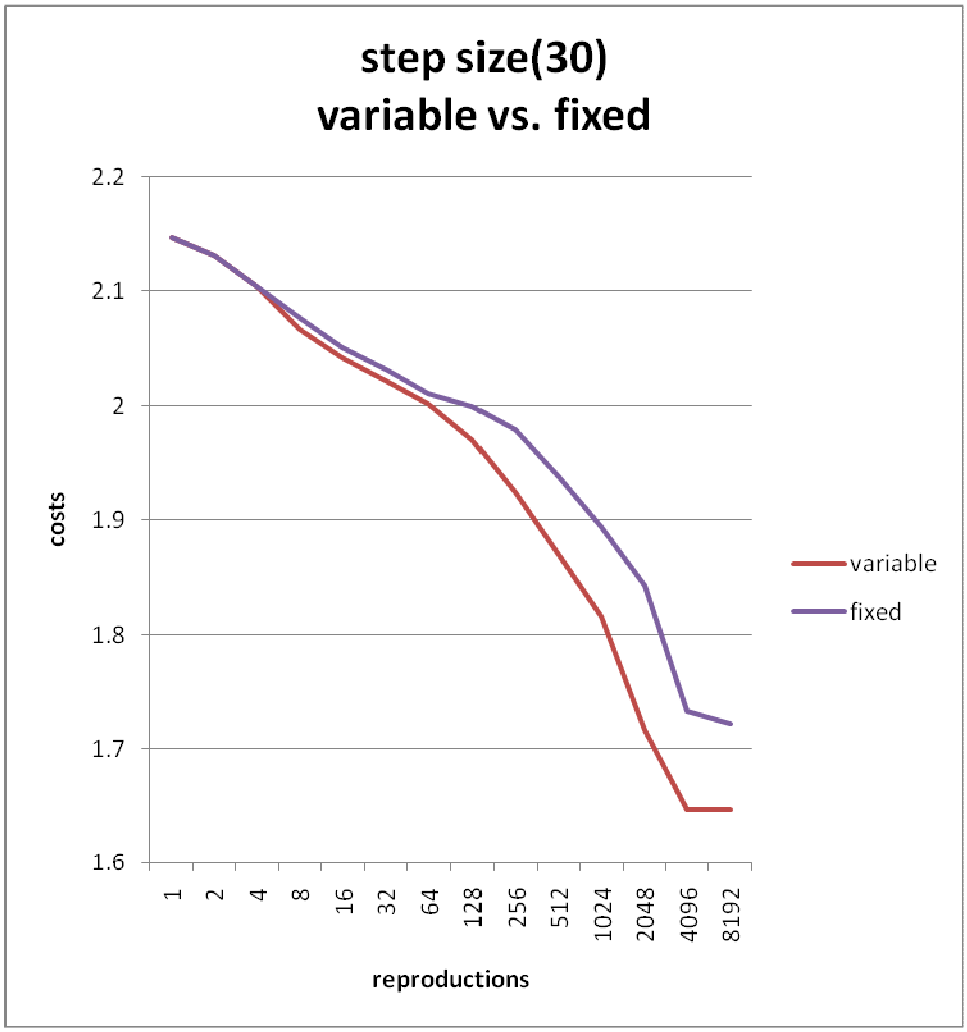


Figure26. Step size (30) variable vs. fixed

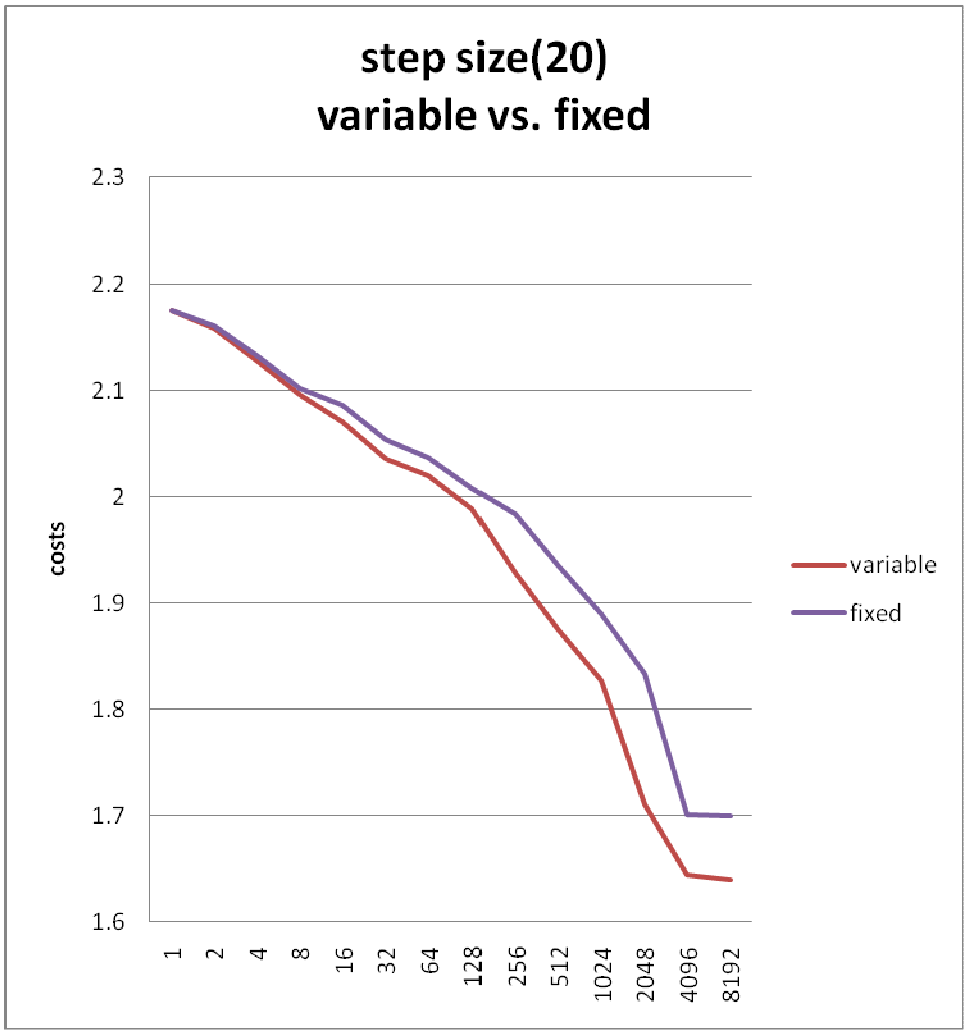


Figure 27. Step size (20) variable vs. fixed

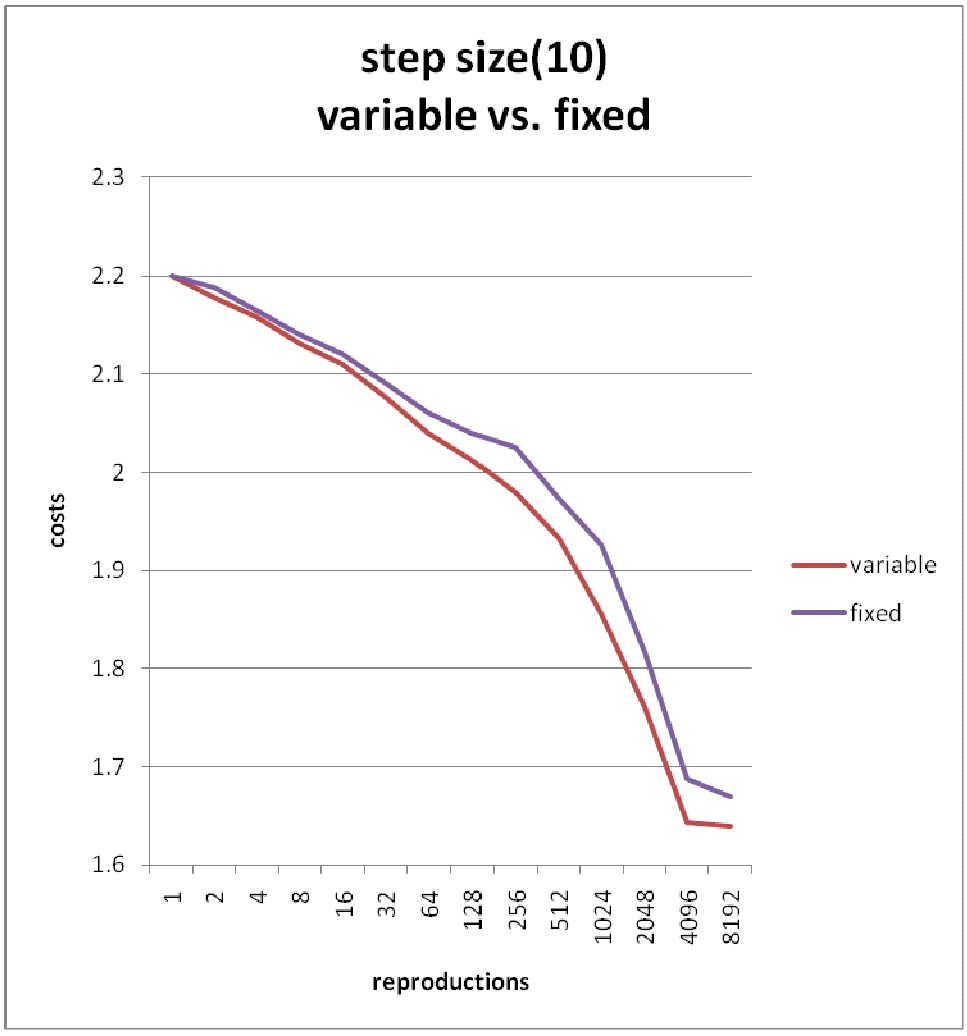


Figure 28. Step size (10) variable vs. fixed

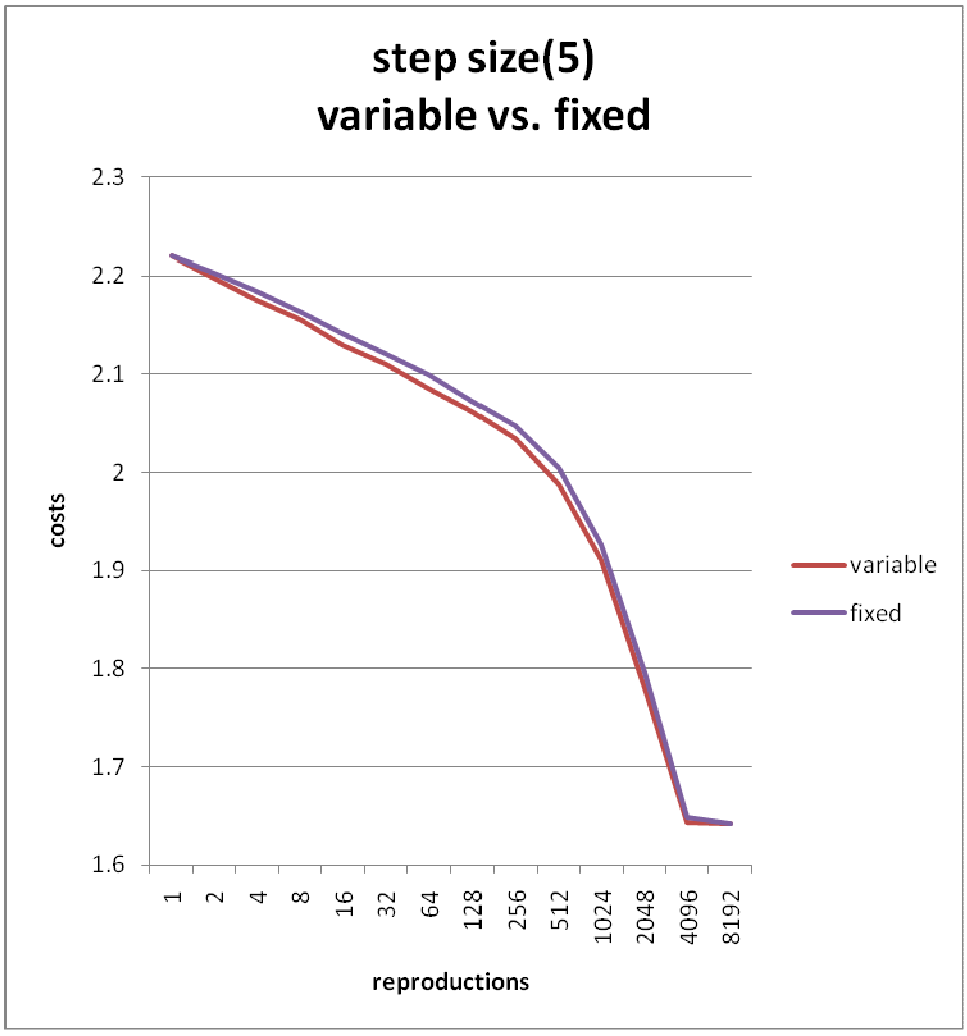


Figure 29 Step size (5) variable vs. fixed

Similar to compare different mutation rates, when we put all the results from four different step sizes, we found at the beginning the bigger the step size the smaller the cost, this is because bigger initial step size can speed up towards the optimal solution. After certain reproductions, there are no big differences for the results. The variable step size scheme works better than the fixed step size scheme, especially when we do not want to wait too

long to converge to the results when the access pattern of a large topology suddenly changes we can start with larger step size and decrease the step size as the genetic operation go through. See figure 30.

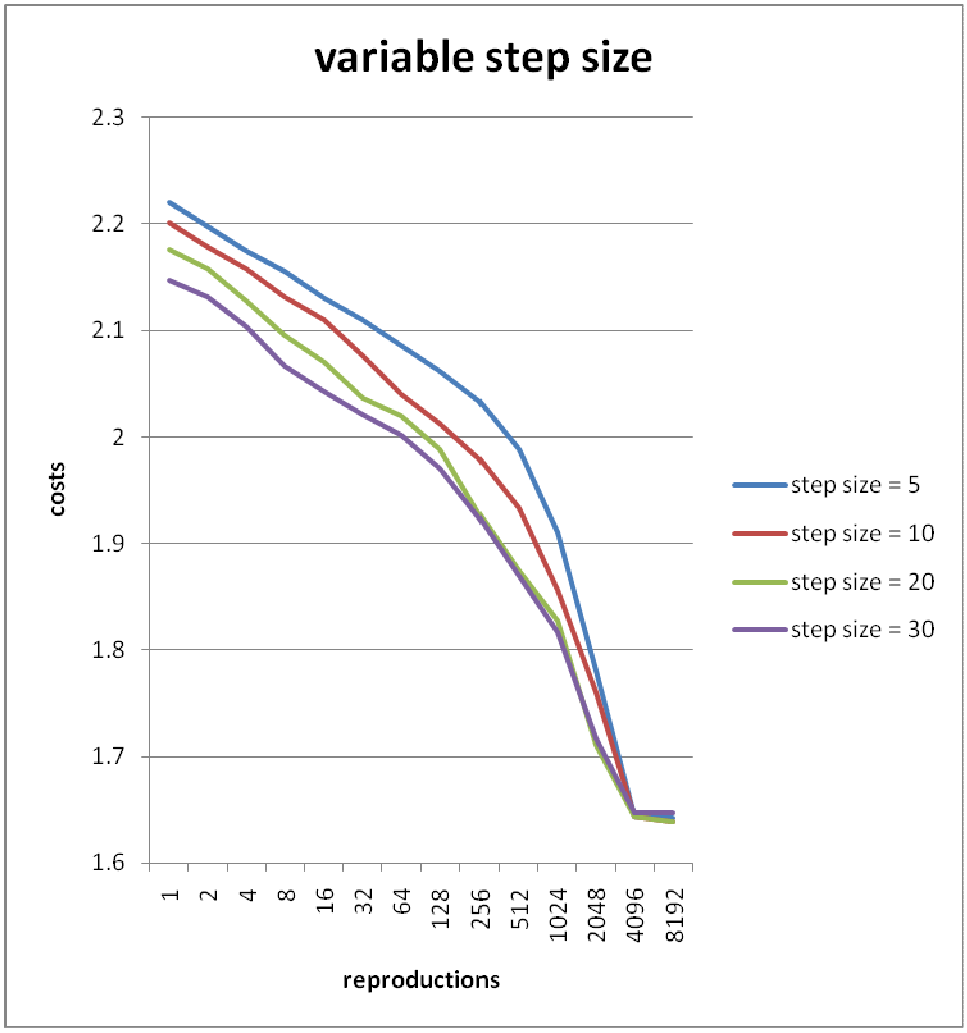


Figure 30. Variable step sizes

9.4 Initial population

The initial population size is the initial number of random replica proxy placements that are created when the algorithm starts. A large population takes longer to find an optimal solution. A smaller population increases the chance that every replica placement in the population will eventually look the same. This increases the chance that the optimal solution will not be found. In our experiments, first we will fix the mutation rate (0.3) and step size (30) and then experiment initial populations with variable mutation rates and step sizes.

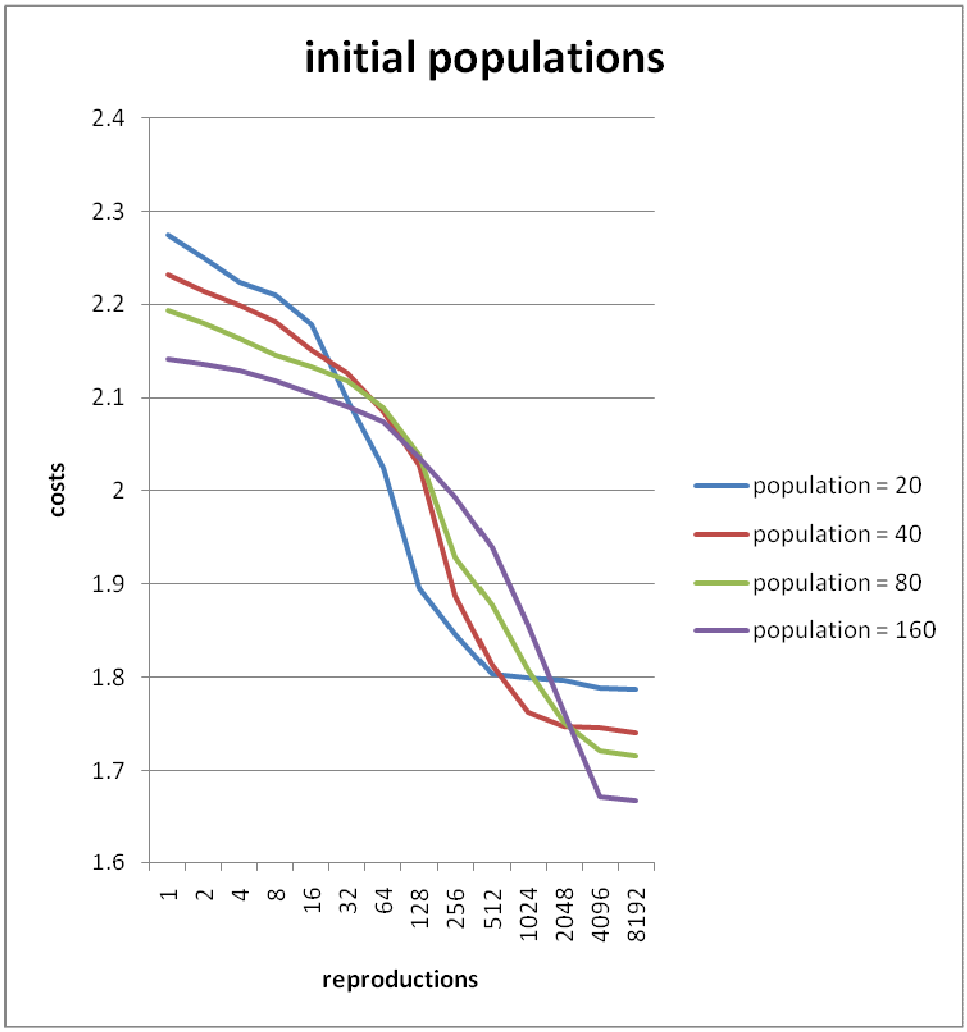


Figure 31 Genetic algorithm parameter sensitivity: initial populations

From Figure 31 we can see for the first few generations, the bigger the initial populations, the smaller the cost, that's because the bigger the initial populations, the more choices we have, the more chance we find the replica placements with smaller cost. After few generations, the experiments with less initial populations converge towards optimal solutions faster than those with more initial populations. After sufficient long generations, the

experiments with more initial populations find the better solution. (smaller costs for the replica placements) The conclusion is that if we can have relatively long time to run the genetic algorithm, we tend to choose relatively more initial populations.

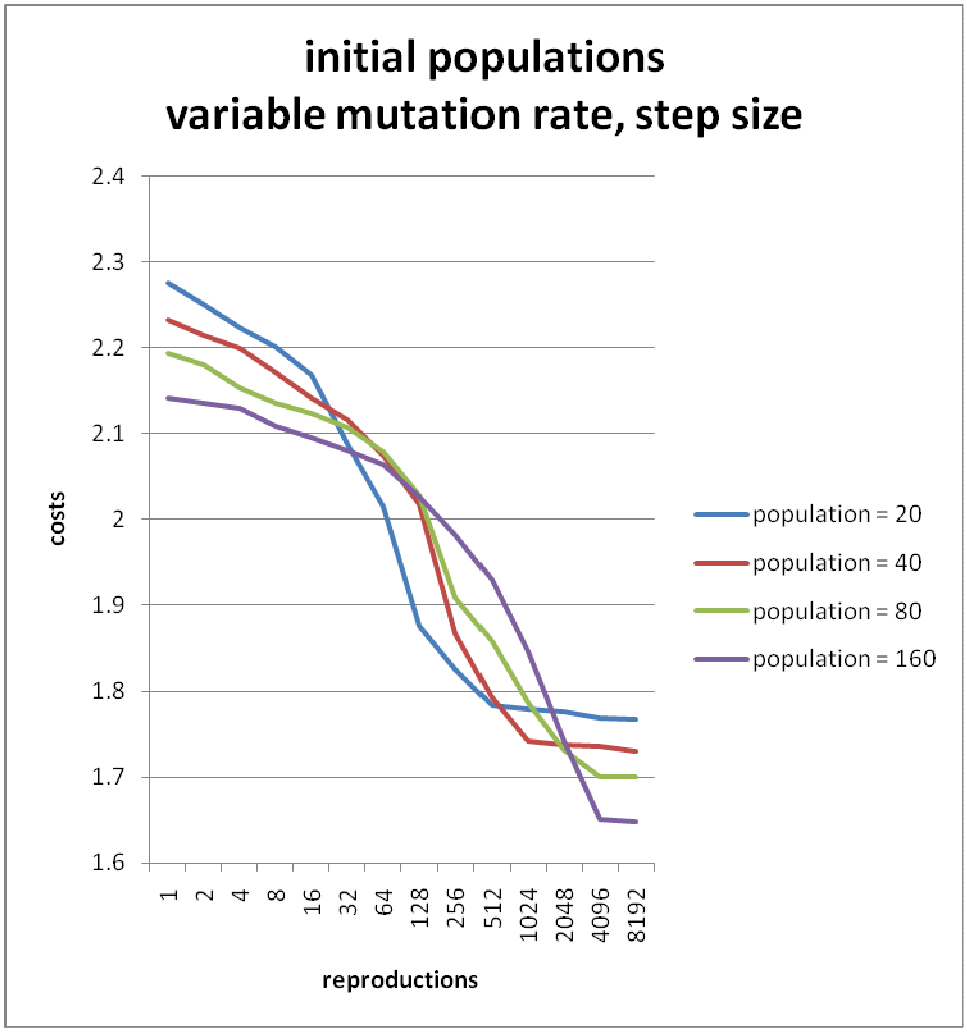


Figure 32: Experiment of initial populations with variable mutation rate and step size.

Figure 32 shows the behavior of the genetic algorithm under different initial populations. The initial mutation rate will be 0.3, the initial step size

will be 30. The mutation rate and the step size will be cut in half when the genetic operations go through half reproductions, mutation rate and step size decrease to one fourth when genetic operations go three fourth. Figure 31 is very similar to figure 31. The only difference is that for each initial population, the cost is smaller than fixed mutation rate and step size, all other features are the same as fixed mutation rate and step size.

In genetic algorithm, we do not have "correct" answer for the genetic parameters. To optimize the genetic algorithm in the context of proxy placement, we varied genetic parameters in order to understand sensitivities in the genetic algorithm results. We experimented with the mutation rate, mutation step size, and initial populations. In the experiments we found initially higher mutation rates appear to speed up the process of finding better proxy placement. However, after certain point, higher mutation rate perturbs the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies. If we change the mutation rate as we go through the genetic operation, we choose higher mutation rate at the beginning of the genetic operation and decrease the mutation rate as we go through the genetic operation, the results are much better. At the beginning we need bigger mutation rate to speed up finding the better proxy placements, as more and more better proxy placements have been found we

do not need to dramatically change the proxy placement, we only need to fine tune the genetic operation, change few proxy site to find the optimal placement. Similar results for the experiment with mutation step size. The variable step size scheme works better than the fixed step size scheme. We can choose bigger mutation step size at the beginning of the genetic algorithm, as we go through the genetic operation we gradually decrease the step size, it will converge to better proxy placement. As for the initial populations our results from the experiments are: large populations take longer to converge to good solutions, although smaller populations converge faster to the solution, the final results are not as good as with large initial populations. So if we want quickly respond to the sudden change of client access pattern we can choose smaller initial population, if we have sufficient response time especially in initial setting up the network we should choose larger initial populations.

10. Simulation results for more topologies

So far we have experiment a small topology and a realistically large topology. Here we will experiment different topologies with sizes in between and compare the results between genetic algorithm, greed algorithm and the optimal solution.

The first network has 28 nodes, 12 clients. The traffic pattern is randomly generated and fixed throughout the simulation. In order to calculate the normalized cost, each request rate is normalized. As for the program parameter we set the initial population as 50 and use variable mutation rate and step size. Initially the mutation rate is 0.40 as the cost change decrease we will change mutation rate to 0.20, 0.10 and 0.05 accordingly. Since the topology size is not that big we start the step size with 4 and later change to 2 and 1 subsequently.

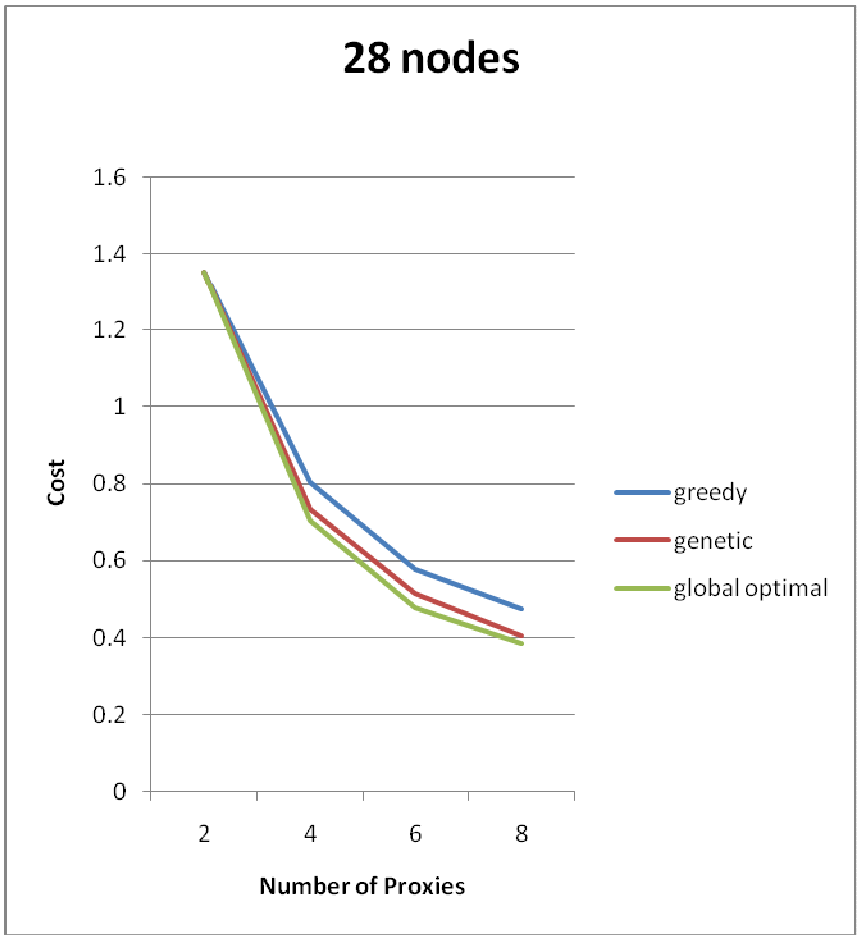


Figure 33. comparison of genetic, greedy algorithm and global optimal (1)

The experiment results are plotted in Figure 33. For small proxy numbers, the results from greedy algorithm, genetic algorithm are very close to global optimal. As the proxy number increase, the genetic algorithm gradually outperform the greedy algorithm, although the results from the genetic algorithm is not exactly same as the global optimal, it is very close to.

The second network has 55 nodes, 30 clients. The traffic pattern is randomly generated and fixed throughout the simulation. In order to calculate the normalized cost, each request rate is normalized. As for the program parameter we set the initial population as 100 and use variable mutation rate and step size. Initially the mutation rate is 0.40 as the cost change decrease we will change mutation rate to 0.20, 0.10 and 0.05 accordingly. We start the step size with 8 and later change to 4, 2 and 1 subsequently.

The experiment results are plotted in Figure 34. For small proxy numbers (5% nodes are proxies), the results from greedy algorithm, genetic algorithm are very close to global optimal. As the proxy number increase,

the genetic algorithm gradually outperform the greedy algorithm, and the results from the genetic algorithm is close to the global optimal.

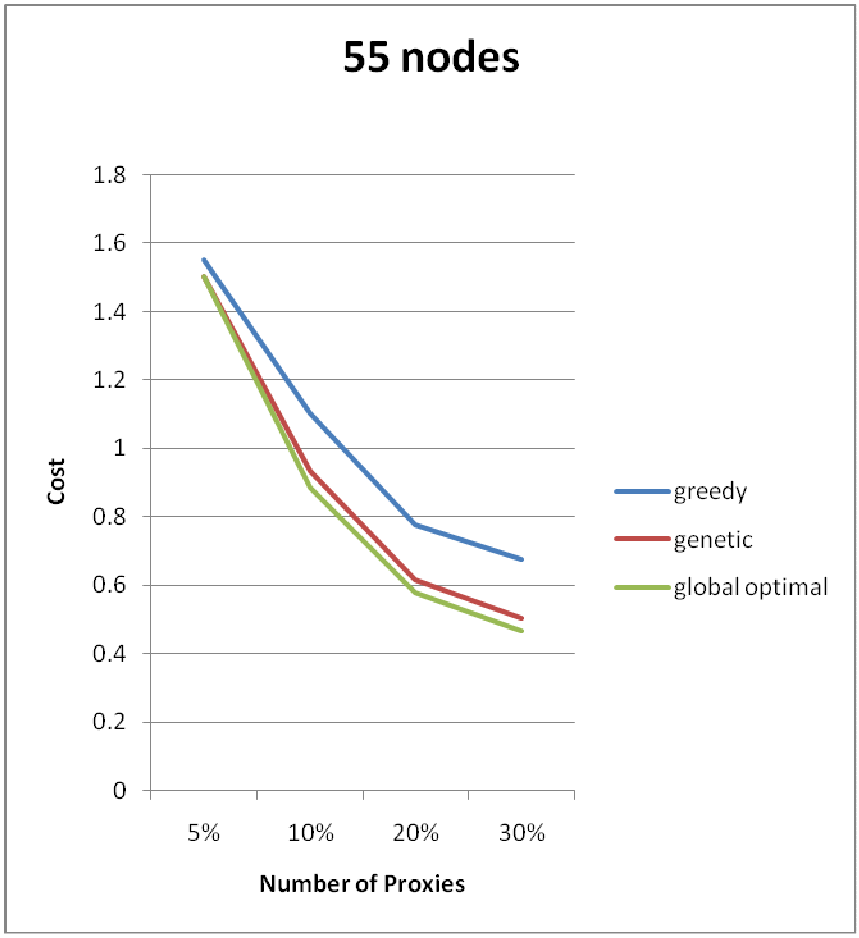


Figure 34. comparison of genetic, greedy algorithm and global optimal (2)

The third network example has 103 nodes, 62 clients. The traffic pattern is randomly generated and fixed throughout the simulation. In order to calculate the normalized cost, each request rate is normalized. As for the

program parameter we set the initial population as 150 and use variable mutation rate and step size. Initially the mutation rate is 0.40 as the cost change decrease we will change mutation rate to 0.20, 0.10 and 0.05 accordingly. We start the step size with 16 and later change to 8, 4, 2 and 1 subsequently.

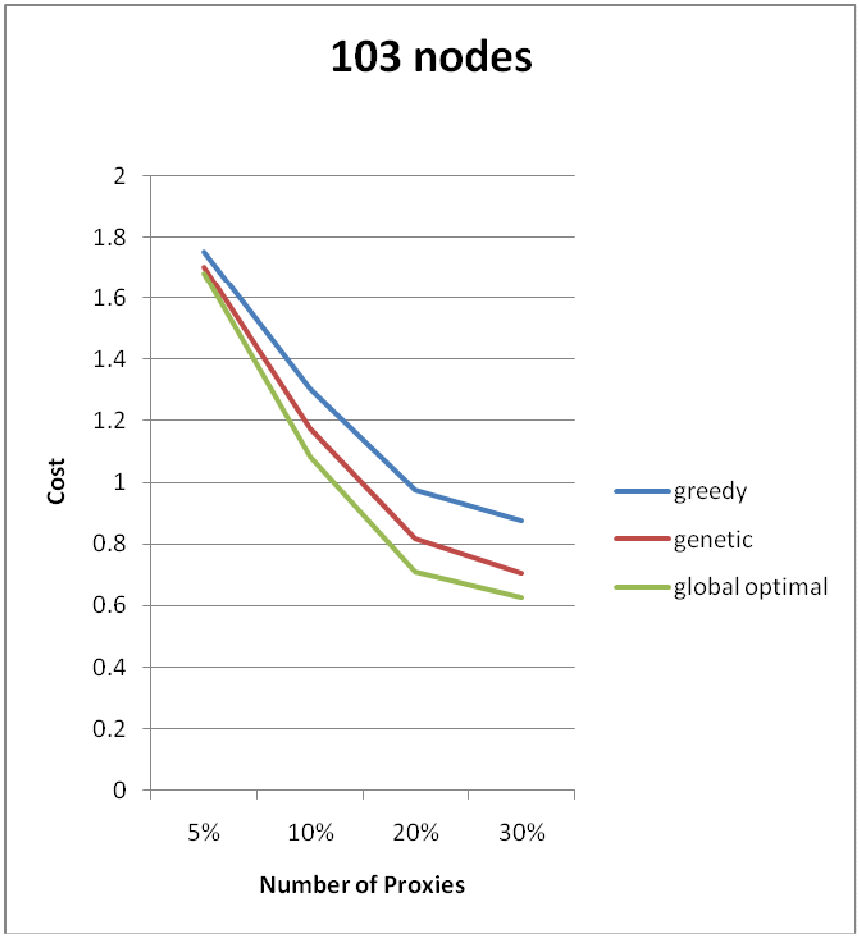


Figure 35. comparison of genetic, greedy algorithm and global optimal (3)

The experiment results are plotted in Figure 35. Once again, similar to the previous two network example, for small proxy numbers (5% nodes are proxies), the results from greedy algorithm, genetic algorithm are very close to global optimal. As the proxy number increase, the genetic algorithm gradually outperform the greedy algorithm, and the results from the genetic algorithm is close to the global optimal.

11. Conclusion

Web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. One of the solutions is to using content distribution network (CDN). When we optimize proxy placement in content distribution network we want to maximize the user experience in the mean time to minimize the resource consumption. These two objects are conflict with each other. To maximize the user experience we need to place proxy as many as possible, but that will increase the resource consumption. To minimize the resource consumption we need to minimize the number of proxy placement and that will increase the user latency so the user experience will decrease. We are dealing with a multi-objective optimization problem here.

After the topology model for content distribution network, we listed metrics to evaluate content distribution network. Then we defined the objective function for optimizing proxy placement. Later we use the objective function to evaluate the effectiveness of a replica placement. After reviewing the existing proxy placement algorithms we used genetic algorithm to solve the Web proxy placement problem.

The greedy algorithm has the advantage of being scalable to larger networks and efficient. Especially when the response time constrain is more important. The drawback of this algorithm is that it cannot guarantee to get the global optimum. It is easy to trap in the local optimum. While the exhaustive search approach to the proxy placement problem guarantees to find an optimal solution for a given topology and client request rates (since all possible combinations has been searched), as the size of the network and the number of proxies grow, this approach is not computationally feasible. The genetic algorithm approach in the results for the simple network matches the results from the exhaustive search, indicating that the objective function represents the feature of network.

The genetic algorithm approach works well not only in the small topology but also in larger topologies. Although for these networks the solution may not be the optimal. Robust good solutions are good enough

compare to optimal solutions. The genetic algorithm can quickly converge to good solutions for the proxy placement problem.

To optimize the genetic algorithm in the context of proxy placement, we found initially higher mutation rates appear to speed up the process of finding better proxy placement. However, after certain point, higher mutation rate perturbs the genetic algorithm, preventing the population from quickly converge to the optimal set of proxies. If we change the mutation rate as we go through the genetic operation, we choose higher mutation rate at the beginning of the genetic operation and decrease the mutation rate as we go through the genetic operation, the results are much better. Similar for the step size if we choose bigger mutation step size at the beginning of the genetic algorithm, as we go through the genetic operation we gradually decrease the step size, it will converge to better proxy placement. As for the initial populations our results from the experiments are: large populations take longer to converge to good solutions, although smaller populations converge faster to the solution, the final results are not as good as with large initial populations. So if we want quickly respond to the sudden change of client access pattern we can choose smaller initial population, if we have sufficient response time especially in initial setting up the network we should choose larger initial populations.

In addition to a simple network and a realistic large network, we experiment three different network examples with their size between the simple network and large network. Since the network sizes are smaller we are able to run the exhaustive search to find the global optimal and compare with the results found in greedy algorithm and genetic algorithm. The results are as expected. When small percentage (5%) of the nodes chosen as proxies, the costs from greedy algorithm, genetic algorithm and the global optimal are very close to each other. As the proxy number increase, the results from genetic algorithm gradually outperform greed algorithm. Although it is not guarantee to find the global optimal solution as the simple network did, a robust good solution is enough for solve the real world problem.

References

[1] Alguliev, R. Alyguliev, R. and Sharifov, M, Model of Optimum Placement of Servers and Web-Contents in Content Delivery Systems *ISSN 0146-4116, Automatic Control and Computer Sciences, 2006, Vol. 40, No. 4, pp. 28–34. © Allerton Press, Inc., 2006.*

[2] Alqaralleh, B., Wang, C., Zhou, B. and Zomaya, A., Effects of Replica Placement Algorithms on Performance of structured Overlay Networks *2007 IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, USA, March 26-March 30*

[3] Bartolini, N., Lo Prestiz, F., Petrioli, C., Optimal Dynamic Replica Placement in Content Delivery Networks, *The 11th IEEE Conference on Networks 2003, ICON 2003*

[4] Buchholz, S., Buchholz, T., Replica Placement in Adaptive Content Distribution Networks *ACM SIG Symposium on Applied Computing. Nicosia, Cyprus; 14--17 March, 2004.*

[5] Bhulai, S., Mei, R., and Wu, M., Heuristics for the Design and Optimization of Streaming Content Distribution Networks *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE Nov 26-30, 2007*

[6]Carhill, A., Sreenan, C., An Efficient CDN Placement Algorithm for High Quality TV Content, *Internet and Multimedia Systems and \ Applications, EuroIMSA 2005, Grindelwald, Switzerland, February 21-23, 2005 364-369*

[7] Chen, Y., Katz, R. and Kubiatowicz, J., 2002a, Dynamic proxy placement for scalable content delivery. *In Proc. 1st International Workshop on Peer-to-Peer Systems (Cambridge, MA). Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 306–318.*

[8] Chen, Y., Qiu, L., Chen, W., Nguyen, L. AND Katz, R. H. 2002b. Clustering web content for efficient replication. *In Proc. 10th International Conference on Network Protocols (Paris, France). IEEE Computer Society Press, Los Alamitos, CA.*

[9] Chen, M., Lapaugh, A., Singh, J. Content Distribution for Publish/Subscribe Services *In Proceedings of the International Middleware Conference. (2003)*

[10] Chuang, Sirbu, 2000 Distributed Network Storage Service with Quality-of-Service Guarantees *Journal of Network and its application* 23(03) 163-185.

[11] Cohen, E. and Kaplan, H. 2001. Proactive caching of DNS records: Addressing a performance bottleneck. In *Proc. 1st Symposium on Applications and the Internet* (San Diego, CA). IEEE Computer Society Press, Los Alamitos, CA.

[12] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. 2002. Globally distributed content delivery. *IEEE Internet Computing 6*, 5 (Sept.), 50–58.

[13] Da Cunha, C. R. 1997. Trace analysis and its applications to performance enhancements of distributed information systems. *Ph.D. dissertation, Boston University, Boston, Mass*.

[14] Dykes, S. G., Robbins, K. A. and Jeffrey, C. L. 2000. An empirical evaluation of client-side server selection. In *Proc. 19th INFOCOM Conference* (Tel Aviv, Israel). IEEE Computer Society Press, Los Alamitos, CA. 1361–1370.

[15] Erman, j., Gerber, A, Hajiaghayi, M., Pei, D., Spatscheck, O. Network-Aware Forward Caching *18th International World Wide Web Conference, Madrid, Spain, April 20-24, 2009*

[16] G. WINTER, J.PERIAUX, M.GALAN, P.CUESTA: Genetic Algorithms in Engineering and Computer Science. *John Wiley & Sons,* 1995

[17] Guo, J., Jha, S. Placing Multicast Proxies for Internet Live Media Streaming *32nd Annual IEEE Conference on Local Computer Networks (LCN 2007), 15-18 October 2007, Clontarf Castle, Dublin, Ireland, Proceedings. IEEE Computer Society 2007*

[18] Haghighat, A., Faez, K., Dehghan, M., Mowlaei, A. and Ghahremani,Y. GA-based heuristic algorithms for bandwidth-delay-constrained least-cost multicast routing. In *Computer Communications, Volume 24, Issues 7-8, 1 April 2001, Pages 685-692*

[19] Huffaker, B., Fomenkov, M., Plummer, D. J., Moore, D. and Claffy, K. 2002. Distance metrics in the internet. In *Proc. International Telecommunications Symposium* (Natal RN, Brazil). IEEE Computer Society Press, Los Alamitos, CA.

[20] Jia, X., Li, D., Du, H. Cao, J. On optimal replication of data object at hierarchical and transparent web proxies *IEEE Transactions on Parallel and Distributed Systems Issue Date: Aug. 2005 Volume: 16 Issue: 8 On page(s): 673 - 685*

[21] Karlsson, M., Karamanolis, C. and Mahalingam, M. 2002 A Framework for Evaluating Proxy Placement Algorithms. Tech. rep., HP Laboratories, Palo Alto, CA.

[22] Karlsson, M. and Karamanolis, C. 2004. Choosing proxy placement heuristics for wide-area systems. In *Proc. 24th International Conference on Distributed Computing Systems* (Tokyo, Japan). IEEE Computer Society Press, Los Alamitos, CA.

[23] Kangasharju, J., Roberts, J. and Ross, K. 2001a. Object replication strategies in content distribution networks. In *Proc. 6th web Caching Workshop* (Boston, MA). North-Holland, Amsterdam, the Netherlands.

[24] Lai, K. and Baker, M. 1999. Measuring Bandwidth. In *Proc. 18th INFOCOM Conference* (New York, N.Y.). IEEE Computer Society Press, Los Alamitos, CA. 235–245.

[25] Li, B., Golin, M. J., Italiano, G. F., and Deng, X. 1999. On the optimal placement of web proxies in the internet. In *Proc. 18th INFOCOM*

Conference (New York, N.Y.). IEEE Computer Society Press, Los Alamitos, CA. 1282–1290.

[26] Lu, l., Li, H., Study on an improved genetic algorithm for facility location problem in forward and reverse logistics**.** *In 2010 International Conference on Computer Application and System Modeling (ICCASM).*

[27] McManus, P. R. 1999. A passive system for server selection within mirrored resource environments using AS path length heuristics. *White paper. Applied Theory, Inc., June.*

[28] Mohammad, S. R., Prashant, J.S., GOYAL, P., Ramamritham, K. Implications of proxy caching for provisioning networks and servers. *SIGMETRICS 2000*: 66-77

[29] PIERRE, G. AND VAN STEEN, M. 2001. Globule: A platform for self-replicating web documents. In *Proc. Protocols for Multimedia Systems*. Lecture Notes on Computer Science, vol. 2213. Springer-Verlag, Berlin, Germany, 1–11.

[30] Pansiot, J. and Grad, D. 1998. On routes and multicast trees in the internet. *ACM Compute Communication Rev. 28*, 1, 41–50.

[31] Presti, F., Petrioli, C., Vicari, C. Dynamic replica placement in content delivery networks *13th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2005.*

[32] Qiu, L., Padmanabhan, V., and Voelker, G. 2001. On the placement of web server replicas. In *Proc. 20th INFOCOM Conference* (Anchorage, AK). IEEE Computer Society Press, Los Alamitos, CA. 1587–1596.

[33] RABINOVICH, M. AND AGGARWAL, A. 1999. Radar: A scalable architecture for a global web hosting service. *Compute Network 31*, 11–16, 1545–1561.

[34] Rabby, M., Ravindran, K., Wu, J. Distributed adaptation algorithms for rate-controlled video multicast over shared infrastructure networks *2010 Second International Conference on Communication Systems and*

Networks (COMSNETS) Bangalore, India

[35] Radoslavov, P., Govindan, R., and Estrin, D. 2001.Topology-informed internet proxy placement. In *Proc. 6th web Caching Workshop* (Boston, MA). North-Holland, Amsterdam, the Netherlands.

[36] Ravindran, K., Wu, J. 2006 'Dynamic protocol plug-in': a Middleware Provision for Enhancing Network Service Performance *First International Conference on Communication System Software and Middleware, 2006. Comsware 2006. New Delhi, India*

[37] Ravindran, K. Wu, J. Performance Management Protocols for Adaptive Content Distribution Networks *International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications Orlando, FL - June 15, 2005*

[38] Ravindran, K., Wu, J. Architecture for Dynamic Protocol-level Adaptation to Enhance Network Service Performance *10th IEEE/IFIP Network Operations and Management Symposium, 2006.*

[39] RODRIGUEZ, P. AND SIBAL, S. 2000. SPREAD: Scalable platform for reliable and efficient automated distribution. *Computer Network 33*, 1–6, 33–46.

[40] Shmoys, D., Tardos, E. and Aardal, K. 1997. Approximation algorithms for facility location problems. In *Proc. 29th Symposium on Theory of Computing* (El Paso, TX). ACM, New York, NY, 265–274.

[41] Sivasubramanian, S., Szymaniak, M., Pierre, G. and Van Steen, M. Replication for web Hosting Systems in *2004ACM Computing Surveys, Vol. 36, No. 3, September 2004,* pp. 291–334.

[42] Sivasubramanian, S., Pierre, G. and Van Steen, M. 2003. A case for dynamic selection of replication and caching strategies. In *Proc. 8th Web Caching Workshop* (Hawthorne, NY).

[43] Tang, X., Xu, J. On Replica Placement for QoS-Aware Content Distribution, *INFOCOM 2004. Twenty-third Annual Joint Conference*

of the IEEE Computer and Communications Societies

[44] Ting, C., Hong, M., On The Use of Genetic Algorithms to Solve the Dynamic Location Problem. In *Journal of the Eastern Asia Society for Transportation Studies, Vol.5, October, 2003*

[45] Triantafillou, P., Aekaterinidis, I. Guiding web Proxy and Server Placement for High Performance Internet Content Delivery, *Web Information Systems Engineering - WISE 2010 - 11th International Conference, Hong Kong, China, December 12-14, 2010. Proceedings Springer 2010*

[46] Unger, O., Cidon, I., Optimal Content Location in Multicast Based Overlay Networks with Content Updates *World Wide Web archive Volume 7 Issue 3, September 2004, Kluwe r Academic Publishers Hingham, MA, USA*

[47] Varadarajan, S., Harinath, R., Srivastava, J. and Zhang, Z Coverage-aware proxy placement for dynamic content management over the Internet, *International Workshop on New Advances of web Server and Proxy Technologies Providence, Rhode Island, USA, May 19, 2003*

[48] Wang, Z., Shi, B., Zhao, E., Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm in *Computer Communications, Volume 24, Issues 7-8, 1 April 2001, Pages 685-692*

[49] Wu, J., Ravindran, K. Optimization Algorithms for Proxy Server Placement in Content Distribution Networks *International Symposium on Integrated Network Management-Workshops, 2009. IM '09. IFIP/IEEE*

[50] Xu, Z., Bhuyan, L., QoS-aware object replica placement in CDNs *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*

[51]Yang, M., Fei, Z. A model for replica placement in content distribution networks for multimedia applications, in: *ICC 2003 - IEEE International Conference on Communications, May 2003.*

[52] Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A. and

Kubiatowicz, J. 2004. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J. Sel. Areas Commun. 22*, 1 (Jan.), 41–53.

[53] Zari, M., Saiedian, H. and Naeem, M. 2001. Understanding and reducing web delays. *Computer 34*, 12 (Dec.), 30–37.

[54] Zhou, X., Lu, X., Hou, M., Wu, J., A dynamic distributed proxy management mechanism based on accessing frequency detecting. *Operating Systems Review 38*(3): 26-34 (2004)