IGP Traffic Engineering: A comparison of Computational Optimization Algorithms

Hong Feng Wang

Thesis presented in partial fulfilment
of the requirements for the degree of
Master of Science
at the University of Stellenbosch

Supervisor: Dr. Antoine B. Bagula
Co-supervisor: Prof. A. E. Krzesinski
March 2008

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: ..... Date: .....

Abstract

Traffic Engineering (TE) is intended to be used in next generation IP networks to optimize the usage of network resources by effecting QoS agreements between the traffic offered to the network and the available network resources. TE is currently performed by the IP community using three methods including (1) IGP TE using connectionless routing optimization (2) MPLS TE using connection-oriented routing optimization and (3) Hybrid TE combining IGP TE with MPLS TE. MPLS has won the battle of the core of the Internet and is making its way into metro, access and even some private networks. However, emerging provider practices are revealing the relevance of using IGP TE in hybrid TE models where IGP TE is combined with MPLS TE to optimize IP routing. This is done by either optimizing IGP routing while setting a few number of MPLS tunnels in the network or optimizing the management of MPLS tunnels to allow growth for the IGP traffic or optimizing both IGP and MPLS routing in a hybrid IGP+MPLS setting.

The focus of this thesis is on IGP TE using heuristic algorithms borrowed from the computational intelligence research field. We present four classes of algorithms for Maximum Link Utilization (MLU) minimization. These include Genetic Algorithm (GA), Gene Expression Programming (GEP), Ant Colony Optimization (ACO), and Simulated Annealing (SA). We use these algorithms to compute a set of optimal link weights to achieve IGP TE in different settings where a set of test networks representing Europe, USA, Africa and China are used. Using NS simulation, we compare the performance of these algorithms on the test networks with various traffic profiles.

Opsomming

Verkeersingenieurswese (VI) is aangedui vir gebruik in volgende generasie IP netwerke vir die gebruiksoptimering van netwerkbronne deur die daarstelling van kwaliteit van diens ooreenkomste tussen die verkeersaanbod vir die netwerk en die beskikbare netwerkbronne. VI word huidiglik algemeen bewerkstellig deur drie metodes, insluitend (1) IGP VI gebruikmakend van verbindingslose roete-optimering, (2) MPLS VI gebruikmakend van verbindingsvaste roete-optimering en (3) hibriede VI wat IGP VI en MPLS VI kombineer. MPLS is die mees algemene, en word ook aangewend in metro, toegang en selfs sommige privaatnetwerke. Nuwe verskaffer-praktyke toon egter die relevansie van die gebruik van IGP VI in hibriede VI modelle, waar IGP VI gekombineer word met MPLS VI om IP roetering te optimeer. Dit word gedoen deur òf optimering van IGP roetering terwyl 'n paar MPLS tonnels in die netwerk gestel word, òf optimering van die bestuur van MPLS tonnels om toe te laat vir groei in die IGP verkeer òf die optimering van beide IGP en MPLS roetering in 'n hibriede IGP en MPLS situasie.

Die fokus van hierdie tesis is op IGP VI gebruikmakend van heuristieke algoritmes wat ontleen word vanuit die berekeningsintelligensie navorsingsveld. Ons beskou vier klasse van algoritmes vir Maksimum Verbindingsgebruik (MVG) minimering. Dit sluit in genetiese algoritmes, geen-uitdrukkingsprogrammering, mierkoloniemaksimering and gesimuleerde temperoptimering. Ons gebruik hierdie algoritmes om 'n versameling optimale verbindingsgewigte te bereken om IGP VI te bereik in verskillende situasies, waar 'n versameling toetsnetwerke gebruik is wat Europa, VSA, Afrika en China verteenwoordig. Gebruikmakende van NS simulasie, vergelyk ons die werkverrigting van hierdie algoritmes op die toetsnetwerke, met verskillende verkeersprofiele.

List of Publications

A.B.Bagula and H.Wang, *On the use of Genetic Algorithms to Fine-tune OSPF Routing*, Southern African Telecommunication Networks Applications Conference (SATNAC), 2005.

A.B.Bagula and H.Wang, *On the Relevance of Using Gene Expression Programming in Destination-based Traffic Engineering*, Lecture Notes in Computer Sciences, Volume 3801, Pages 224-229, December 2005.

Contents

List of Figures

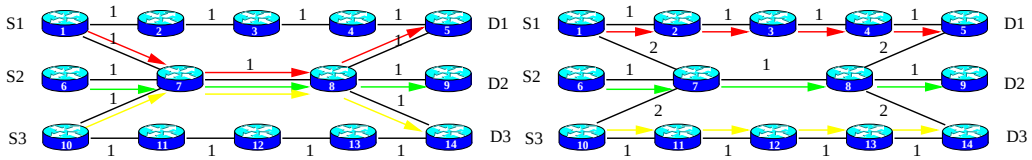
List of Tables

Chapter 1

Introduction

The rapid growth in Internet users and applications has increased the demand for the usage of the Internet resources such as routers, bandwidth, Web servers, etc. This has raised the need for new network management techniques allowing the network resources to match the application requirements in order to deliver the Quality of Service (QoS) demanded by modern IP applications. At the network level, the Internet is divided into routing domains also referred to as Autonomous Systems (ASes) interacting with each others to control and deliver the traffic offered by IP applications. Each of these domains fall under a single institution administration such as a company, a university or an Internet Service Provider. While inter-domain communication is achieved using the Border Gateway Protocol (BGP) [1] to exchange routing information between IP domains, intra-domain communication uses Interior Gateway Protocols (IGPs) such as Open Shortest Path First (OSPF) [2] or Intermediate Systems-Intermediate System (IS-IS) [3] to select the paths used by the traffic within an IP domain. Modern IP network operation practices for intra-domain routing include the assignment of link weights in IGP protocols to route the traffic offered to a network within an IP domain and the calculation and dissemination of link state updates to the edge of an IP domain. Using the complete knowledge of the domain's topology each edge router computes the least cost paths to each destination and creates forwarding tables used to direct each IP packet to the next router on the path to its final destination. The weight values (costs) assigned to the links in OSPF are selected in the range $[1, 65535]$ where $65535 = 2^{16} - 1$.

It is currently recognized that when using traditional destination-based routing, IGP proto-



(a) non-optimal routing (b) optimal routing

Figure 1.1: Example of TE problem

cols lead to unbalanced network configurations where some resources are over utilized while others remain idle. Such unwanted situations can be avoided by using traffic engineering (TE) or network engineering (NE). The objective of TE is to move traffic to where the resources are available in the network to improve routing performance (e.g. minimize link utilization, minimize propagation delay, etc.). In contrast, NE moves the network resources to where the traffic is offered to the network to achieve the same routing objective. Both of these network management techniques involve different optimization algorithms based on exact methods or heuristic methods. The focus of this thesis lies on computational algorithms used to achieve TE.

1.1 The TE problem

Figure 1.1 illustrates two network configurations where three traffic flows are routed on three origin-destination pairs (i.e. S1-D1; S2-D2; S3-D3). Traditional destination-based routing will lead to the non-optimal network configuration which uses the paths $1-7-8-5$ on the S1-D1 pair, $6-7-8-9$ on the S2-D2 pair and $10-7-8-14$ on the S3-D3 pair as illustrated by Figure 1.1 (a). As a result the link $7-8$ used by three flows may become a bottleneck for traffic engineering leading to congestion or higher loss upon failure though the network still has enough resources (bandwidth) to route the three traffic flows without overloading any link of the network. An optimized network configuration can be obtained by separating the three flows to allow each of the flows to be routed on its own path. This can be done by adjusting the weight (cost) on the links $1-7$; $8-5$; $10-7$; and $8-14$ from one to two as depicted by Figure 1.1 (b) while leaving the other link costs to 1. This reflects a situation where TE is used to improve the network performance by manipulating the link weights.

TE is currently performed by the IP community using three methods: (1) Interior Gateway Protocol (IGP) TE using connectionless routing optimization, (2) Multiprotocol Label Switching (MPLS) TE using connection-oriented routing optimization, and (3) Hybrid TE combining IGP TE with MPLS TE. At its outset, MPLS TE raised controversies concerning how the future Internet will be engineered. On one hand, the IP engineers believing that the Internet could be engineered without the need of sophisticated network management such as provided by MPLS, proposed a single and scalable Internet where the routing of the traffic in the Internet could be optimized by optimizing only the IGP link weights. On the other hand, telecommunication operators adopted a connection oriented routing model borrowed from the ATM virtual circuit model referred to as MPLS. While MPLS has won the battle of the core of the Internet and is making its way into metro, access and even some private networks, emerging Internet service provider operation practices are revealing the relevance of using IGP TE in hybrid TE models where IGP TE is combined with MPLS TE to optimize IP networks. This is done by either optimizing IGP routing using the link weight optimization paradigm while setting a few number of MPLS tunnels in the network as proposed by [4] or by optimizing the management of MPLS tunnels to allow growth for the IGP traffic as proposed in [5]. A third option consists of optimizing both IGP and MPLS routing in a hybrid IGP+MPLS setting. The focus of our work lies on IGP TE using heuristic algorithms borrowed from the computational optimization research field.

1.2 The IGP TE Problem

Given a directed acyclic graph $G(\mathcal{N}, \mathcal{L})$ where $\mathcal{N}$ is the set of nodes representing the routers of the network while $\mathcal{L}$ is the set of arcs representing the capacitated network links between the routers. Let $D = (r_{i,e})$ denote a demand matrix where $r_{i,e} = (i, e, d_{i,e})$ is a triple expressing a request to route $d_{i,e}$ units of data traffic between the source i and the destination e.

Problem 1: Routing optimization problem

The routing problem consists of finding a set of paths to route the demand D in order to minimize a measure of congestion defined by the objective function Ω given by:

$$\Omega = \max_{\ell \in \mathcal{L}}(\mu_\ell) \tag{1.1}$$

where $\mu_\ell = \frac{f_\ell}{C_\ell}$ is the utilization of link ℓ, f_ℓ is the total traffic carried by link ℓ and C_ℓ is the capacity of link ℓ. Note that as defined above

- The routing problem above is an optimization problem consisting of minimizing the maximum link utilization.
- The demand matrix D can be estimated using future demands based for example on concrete measures of flow between source-destination pairs as proposed in [6, 7] for the AT&T Worldnet backbone or predicted from a concrete set of consumer subscriptions to virtual leased lines as suggested by [8].

Problem 2: Link weights setting problem
Given the network model defined above and a demand matrix D expressing as above the demand in traffic flow between origin-destination pairs, the routing optimization problem **Problem 1** can be redefined as a link weight setting problem consisting of determining a set of weights $\mathcal{W}=\{w_\ell\}$, where each weight $w_\ell \in [1, Maxwt]$ such that the objective function Ω is minimized.

1.2.1 A hybrid routing architecture

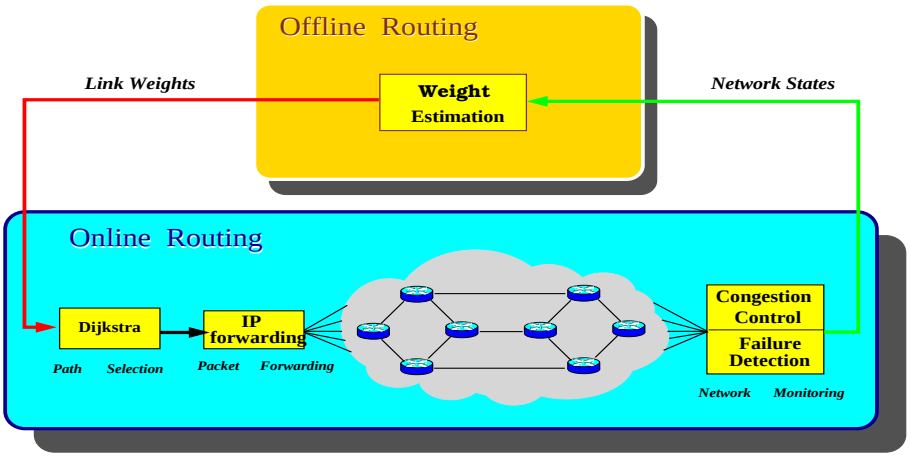


Figure 1.2: Architecture of the hybrid TE model

The IGP TE problem above has been solved using methods borrowed from computational and non-computational intelligence and its solution can be implemented using a hybrid

offline/online routing architecture using offline link weight calculation and online routing by performing path selection and packet forwarding to achieve optimized IGP routing. The architecture depicted by Figure 1.2 reveals a two-layer model where the offline weight calculation is separated from the online packet forwarding. The upper layer of the architecture implements two functions: (1) collecting current network state (e.g, network topology, traffic demands, propagation delays; etc.) and (2) calculating the link weights based on the network state. The lower layer implements three functions: (1) path selection using Dijkstra's algorithm with link costs set to the link weight values (2) populating forwarding tables to use the selected paths and (3) network monitoring through congestion control and failure detection.

Note that these functions are defined by the closed loop of our proposed routing architecture where (1) the path selection in the bottom layer is performed based on the weights computed by the weight selection in the upper level (2) the packet forwarding in the lower layer is defined by the paths selected during path selection and (3) the weight estimation in the upper layer is triggered by the network monitoring engine in the lower layer when new link weights need to be calculated. As proposed in [2], the guideline on how to best design OSPF networks is to use at most 200 routers in a single area/domain. As suggested by CISCO in [9], no more than six router hops from source to destination and 30 to 100 routers should be used per area/domain, but less than 40 routers is recommended to achieve OSPF routing scalability.

1.3 Related work

It was proved in [10] that finding not only the optimal OSPF weight setting but even an approximate solution to the problem was NP-hard. A piece-wise linear and convex function Ω is defined and used in [10] as a congestion measure for the weight setting problem in OSPF routing. This problem is solved using a local search heuristic. Using a single descent and working with small networks of at most 16 nodes and 18 links, a local search procedure similar to [10] is presented in [11] while the work presented in [12] use local search with a single descent in a model which deals simultaneously with the network design problem on small networks with at most 13 links. A completely different approach

using Lagrangian relaxation is presented in [13] to solve the weight setting problem on networks of up to 26 nodes. The work presented in [14] deals with search heuristics for load balancing in IP networks. This work studies three different heuristics proposed in [10] and [11] and evaluate their performance using topologies with power-law properties. The evaluation results revealed that the heuristic proposed by [10] performs better than the other and achieves results which are reasonable close to the optimum but at the price of high processing time.

A solution to the OSPF weight setting problem is proposed in [15]. The paper formulates a relevant OSPF routing optimization problem, proves its NP-completeness, and discusses possible heuristic approaches and related optimization methods for solving it. These heuristics include simulated annealing and genetic algorithms. Two basic approaches to the OSPF weight setting problem are considered in the paper and the resulting optimization algorithms presented: a direct and a two-phase approach. Heuristic solutions using analogies with natural and social systems have been proposed to optimize IGP routing [8, 16, 17, 18, 19]. Building upon the congestion measure in [10], [8, 18, 19] solve the weight setting problem in the OSPF routing using a genetic algorithm in [8] and a hybrid genetic algorithm [18, 19]. In [18, 19], a hybrid genetic algorithm is proposed with a local improvement procedure for the OSPF weight-setting problem. The local improvement procedure makes use of an efficient dynamic shortest path algorithm to recompute shortest paths after the modification of link weights. Using a set of real and synthetic test problems, the model showed near-optimal solutions. In [16], the OSPF weight setting problem is solved by using a local search to complement the global search implemented by classical genetic algorithms to improve the genetic individuals fitness through hill-climbing and speeding up the genetic search. The local search is used to map the link weights to the offered load by diverting traffic from the link with the highest utilization.

A newly developed evolutionary computation method was proposed in [20] under the Gene Expression Programming (GEP) label. While **GA** individuals are symbolic strings of fixed length referred to as "chromosomes", GEP individuals are expressed as non-linear entities of different sizes and shapes referred to as "Expression Trees" consisting of a function of + and terminals as usually expressed by the *Karva* GEP language [20]. GEP was used in [21] as a new approach for solving IGP TE problems and in [22] as a new approach solving both the IGP TE and MPLS TE problems with better performance compared to

basic GA methods because of its richer genetic operations.

Ant-based load balancing was proposed in [23] to achieve load balancing of calls offered to a circuit-switched telecommunication network. The *AntNet* algorithm was proposed in [24, 25] to solve a routing problem closely related to the weight setting problem. It consists of dynamically routing the traffic in packet-switched networks. A recent performance analysis of the *AntNet* algorithm [26] has revealed that the algorithm's performance is comparable to Dijkstra's algorithm but adapts better to varying traffic loads and performs better than shortest path routing.

1.4 Thesis Contribution and Outline

Most of the computational intelligence algorithms proposed in the literature to solve the OSPF weight setting problem have been compared to the OSPF benchmark using link weights which are set inversely proportional to the link weight as suggested by CISCO [27]. To the best of our knowledge, the comparison between these algorithms using different test networks under different traffic conditions has been either scarcely or poorly addressed by the research community. The focus of this thesis lies on IGP TE using computational intelligence algorithms. We present four computational intelligence algorithms and compare their performance when routing the traffic offered to a network with the objective of minimizing the maximum link utilization. These algorithms include Genetic Algorithm (GA), Gene Expression Programming (GEP), Ant Colony Optimization (ACO), and Simulated Annealing (SA). Our work lies in a link weight optimization framework where the four algorithms are used to compute a set of link weights which are used as link costs in IGP routing following the hybrid routing architecture depicted by Figure 1.2. The remainder of this thesis is as follows.

In chapter 2, the IGP TE problem is solved using the classic GA algorithm and its hybrid referred to as hybridGA while chapter 3 solves the same problem using GEP and its hybrid called HybridGEP. Chapter 4 covers extensions to the classic ACO algorithm to solve the IGP TE problem and presents a hybrid ACO algorithm referred to as HybridACO. The Simulated annealing algorithm is presented in chapter 5 in two versions: a classic version referred to as SA and a hybrid version called HybridSA, both used to solve the IGP TE

problem. Chapter 6 compares the hybrid versions of the four algorithms for different networks and traffic conditions. Our conclusions are presented in chapter 7.

Note that

- While IS-IS is based on the same principles as OSPF, we have selected as benchmark for our experimentation the OSPF protocol since, to the best of our knowledge, it is the IGP protocol most used by service providers for intra-domain routing.
- The guidelines for OSPF routing mentioned earlier have been used in this thesis to select the test network models illustrating the main features and used to evaluate the performance of the algorithms used. These include a USA network with 23 nodes and 76 links, a $Europe$ network with 30 nodes and 90 links, an $Africa$ network with 31 nodes and 128 links and a $China$ network with 48 nodes and 145 links.
- The experimental results presented in this thesis are based on different traffic profiles where low traffic profiles are defined by a traffic matrix D where the traffic is offered to a small number of origin-destination pairs while higher traffic profiles are expressed by a traffic matrix with a higher number of origin-destination pairs.
- While simple illustrative experiments are presented in chapter 2 to chapter 5 to compare classic and hybrid versions of the different computational intelligence algorithms under light traffic profiles, chapter 6 presents more robust experimental results where the different computational intelligence algorithms are compared under higher traffic profiles.
- while the use of high traffic profiles may be relevant to evaluate the scalability of the algorithms, IP practices have shown that only a small subset of the origin-destination pairs are responsible for the majority of the traffic carried by the Internet. This validates the illustrations on networks with light traffic profiles used in this thesis work.

Chapter 2

Genetic Algorithms

Genetic algorithms (GAs) are evolutionary algorithms which use concepts from real-world genetics to evolve solutions to problems. They are based on an evolutionary paradigm where each iteration of the algorithm transforms one population of individuals into a new generation, using some pre-determined fitness measure for an individual. The potential solutions found by the evolutionary process are represented and encoded in terms of *genome*. Each problem generally has its own genome representation, and more than one representation could be used for a given problem. A fitness measure or *fitness function* is used to determine how good the solution represented by some genome is and the appropriate fitness function is determined by the problem and genome representation. *Hybrid genetic algorithms* also often referred to as *Memetic Algorithms* belong to a class of evolutionary algorithms where the basic genetic operations are complemented by a local search used to improve the genetic individuals fitness through hill-climbing and speeding up the evolutionary process.

This chapter presents the implementation of a classic GA algorithms named as GA and its hybrid referred to as HybridGA, both solving IGP TE problem. Section 2.1 introduces the classic GA while Section 2.2 presents the application of GA to the IGP TE problem. Section 2.3 presents HybridGA and its application to the IGP TE problem while Section 2.4 illustrates some differences between the two algorithms and the impact of the genetic operations on the performance of the different algorithms. Section 2.5 concludes on GA, Hybrid GA and their application based on the illustrative results.

2.1 Introduction to GA

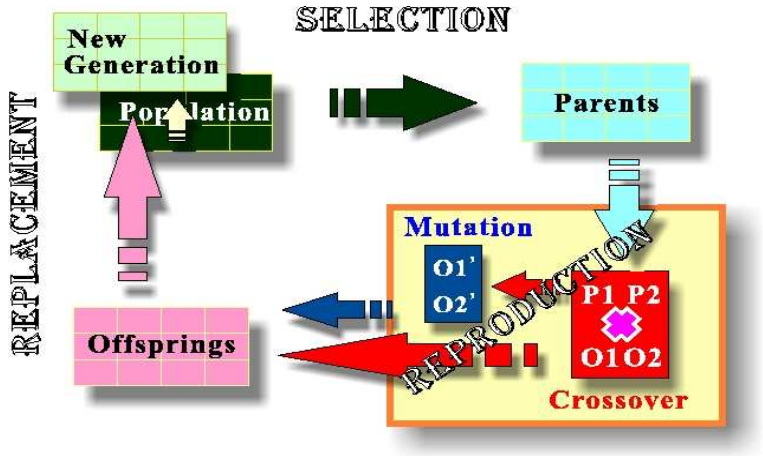


Figure 2.1: Evolutionary loop of GA

GA was first introduced by John Holland in 1975. Genetic Algorithms (GAs) are global optimization techniques which borrow from the principles of natural selection the Darwinian theory of *"survival of the strongest"* where the evolution of individuals is such that "the weakest individuals disappear" while the "strongest individuals survive". Genetic Algorithms (GAs) have proved to be robust search heuristics which are based on three main genetic operations: (1) *replacement* (2) *crossover* and (3) *mutation*. *Replacement* is a direct copying of a member of the current generation into the next generation. *Crossover* is the combination of two genomes from the current generation into two different genomes in the next generation. Crossover attempts to combine good solutions to find potentially better solutions. *Mutation* achieves a random permutation of one of the tokens in the Genome representation of a member of the current generation. By introducing new solutions at each stage of the algorithm, mutation ensures that the evolution process does not get stuck at a local optimum. There are probabilities associated with the crossover, replacement and mutation operations. These probabilities are denoted P_c, P_r and P_m respectively, and $P_c + P_r + P_m = 1$. In general, $P_m \ll P_c$ and $P_c \approx P_r$. While the candidates for the genetic operations are chosen randomly, the selection is *fitness-proportionate* to ensure the survival of good solutions over generations. The conditions for the termination of the algorithm are problem-specific, although for practical reasons one often limits the number of iterations. The genetic operators aim to achieve three goals: inheritance, mutation, and competition. Inheritance transfers gene materials from the parents to the off-springs through crossover.

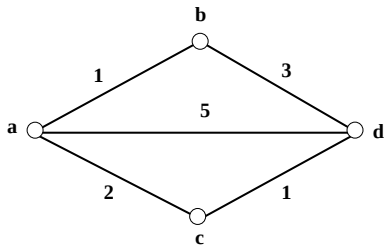
A combination of inherited materials makes the off-springs similar to the parents but could be different from their parents. Mutation simply changes some genes before generating a child. It gives new genes to a child. These genes are different from those inherited from the parents. The competition ensures that an individual with better genes (so better fitness) has more chance to survive. The probabilities assigned to the operators express their frequency of use.

The pseudo-code of the GA algorithm [28] presented below follows the evolutionary loop depicted by 2.1 while its application to the IGP TE OSPF problem is described in section 2.2.

Notation:

g: generation
$C_g = \{\vec{C}_{g,n} | n = 1, \ldots, N\}$: the Population of generation g, where N is the number of individuals of the Population
$O_g = \{\vec{O}_{g,m} | m = 1, \ldots, M\}$: the Offspring generated by generation g, where M is the number of individuals of the Offspring

Algorithm: classic GA
Step1: Set the current generation $g = 0$.
Step2: Initialize the population C_g.
Step3: Evaluate the fitness of each individual $\vec{C}_{g,n} \in C_g$.
Step4: Select parents from C_g.
Step5: Perform cross-over on selected parents to form the offspring O_g.
Step6: Mutate each offspring in O_g.
Step7: Select individuals from C_g and O_g to form new generation C_{g+1} according to their fitness: $C_g = C_g + 1$.
Step8: Stop when a stop condition is met.
Step9: Set $g = g + 1$. **GOTO** step3.



(a) Network Topology

link	ab	ac	ad	ba	bd	ca	cd	da	db	dc
link weight	1	2	5	1	3	2	1	5	3	1

(b) Links and Link Weights

Figure 2.2: A five-link network with assigned link weights

2.2 Application to IGP TE Problem

As formulated in chapter 1, the IGP TE problem can be solved using the genetic evolution by mapping the routing of real traffic in a network into a link weight finding process using genetic operations involving (1) appropriate chromosome representation, (2) mapping the routing objective into a genetic fitness function and (3) using genetic operators to evolve the evolutionary process towards selection of the fittest individuals (best link weights).

2.2.1 Chromosome Representation

In our specific application, a chromosome represents the set of weights assigned to the links of the network. The number of genes in a chromosome will therefore be equal to the number of the links of the network. The **binary** and the **Gray** coding are often used for chromosome encoding. But for convenience, we consider an **Integer** coding where each gene encodes an integer number in the range $[1, 65535]$ as a link weight where the upper bound $65535 = 2^{16} - 1$ reflects current ISP practices. For practical reasons, we have constrained the link weight values in the range of $[1, MaxWT]$, where $MaxWT < 65535$ is the upper bound of the link weight values. In our design, a chromosome is represented by a string of a fixed length integer value where each position is associated to a single link of the network, and the integer in that position is the weight for that single link. For instance, a chromosome representing the set of link weights of the bidirectional five-link network depicted by Figure 2.2 is expressed by the string 1251321531. Since the total number of the links (directional link) is 10, the length of the chromosome is also 10.

2.2.2 Initialization of Population

In our application, the initial Population is randomly generated. Each gene expresses a random integer in the range of [1, $MaxWT$]. Each chromosome has a fix length determined by the number of links. The Population has a certain number of chromosomes. This number must be carefully chosen since it affects the convergence of the algorithm. Obviously, selecting a high number of initial chromosomes leads to a wider search space for each generation and more chances to converge to the global optimum, but at the price of higher processing time. The size of the Population should be set properly for different sizes of the network in order to improve the performance of the algorithm and reduce the processing time. Note also that setting the number of chromosomes to a low value for small networks and higher value for larger networks may improve the convergence of the algorithm.

2.2.3 Fitness Function

We consider a fitness function which is defined by:

$$F_{g,n} = \left(\frac{1}{Objective_{g,n}} \right)^{p_{g,n}} \tag{2.1}$$

where the power $P_{g,n}$ is a sensitivity parameter, and $Objective_{g,n}$ is a penalty function.

Usually, a fitness function expresses either a reward assigned to the fittest individuals when the routing objective is a penalty function to be minimized, or a penalty assigned to the fit-less individuals when the routing objective is a reward function to be maximized. The former applies in our case since we are using as objective the maximum link utilization which is a penalty function. For each individual of Population n and each generation g, the power $P_{g,n}$ can be a constant or a function of both the current and the previous objective values, $P_{g,n} = f(Objective_{g-1,n},\ Objective_{g,n})$. It is used to increase the difference of fitness among individuals of the Population. For example, let

$$p_{g,n} = \begin{cases} 0.5 & \text{if } Objective_{g,n} > Objective_{g-1,n} \\ 1.5 & \text{if } Objective_{g,n} < Objective_{g-1,n} \\ p_{g-1,n} & \text{if } Objective_{g,n} = Objective_{g-1,n} \end{cases}$$

$P_{g,n}$ is either increased ($P_{g,n} = 1.5$) to speed up the convergence when the evolution improves the individuals or decreased ($P_{g,n} = 0.5$) to delay the convergence to avoid that the fit-less solutions disappear and thus lead to premature convergence. Note that the fitness function can be more complex for complex objectives (e.g. multiple objectives, constrained objectives, etc.) The study of these functions is beyond the scope of this thesis.

2.2.4 Genetic Operators

Selection

We use for our implementation of the GA and HybridGA a probabilistic method known as *Roulette Wheel Selection* where the genetic individuals are selected based on their fitness by providing to the fittest individuals a higher probability of being selected. The selection probability is expressed by:

$$Psel_{g,n} = \frac{F_{g,n}}{\sum_{i=1}^{N} F_{g,i}} \tag{2.2}$$

where N is the number of chromosomes in the Population.

Cross-over

Cross-over exchanges some genes between two chromosomes. It helps to enlarge the search space. One-point crossover has been considered for our implementation rather than two-points or multi-points crossover because of its higher potential in chromosome inheritance.

Mutation

Mutation changes some genes of a chromosome. It helps to widen the search space for the next generation in order to avoid that the genetic evolution get stuck to local optima. We consider a mutation where based on probability, each offspring is mutated at a random gene position by using a random integer in $[1,\ MaxWT]$.

2.2.5 The local search

In order to improve the convergence to the global optimum, we consider a hybrid approach where a local search process is combined with the fitness calculation. The idea is, for each population, to search the chromosome leading to local optimum and modify its genes to improve performance. In practice, we implemented the local search process by repeatedly increasing the weight of the link having the maximum link utilization until the maximum link utilization does not decrease further. When integrated to the GA algorithm as depicted by Figure 2.3, this local search process leads to a hybrid genetic algorithm referred to as HybridGA.

2.3 HybridGA Algorithm

Figure 2.3 presents the flowchart of HybridGA. Differing from the classic GA skeleton, the local search process is added to route traffic demands and change the link weights to improve the fitness. Note that in the flowchart, the dices express the randomness related to the operations which are run based on probabilities or use random processes. The HybridGA algorithm is more detailed by the following pseudo-code where are presented (1) a main program (2) the different genetic operations and (3) the local search procedure.

Purpose: Minimizing Maximum Link Utilization.

Notation:

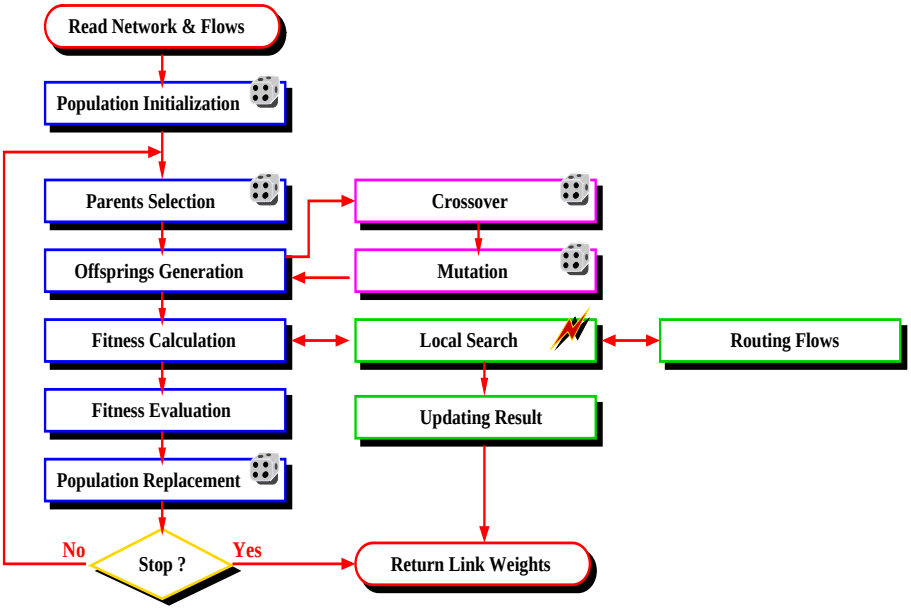


Figure 2.3: Flowchart of HybridGA

Number of nodes: $N \in \mathcal{I}$;
Network nodes: $\mathcal{N}$, $\mathcal{N}$= $\{3, \ldots, N\}$;
Number of links: $NrLink$, $1 \leq NrLink \leq N * (N-1)$;
Link: $link_{i,j}$, $i, j \in \mathcal{N}$;
Links: $\mathcal{L}$= $\{l_k | k = \{1, \ldots, NrLink\}\}$;
Link Capacities: $\mathcal{C} = \{c_{i,j} | c_{i,j} \in \mathcal{R};\ i, j \in \mathcal{N}\}$;
Generation: g;
Number of Generation: G;
Number of chromosomes in the Population: $POPU$;
Upper bound of link weight value: $MaxWT$;
Link weights: $\mathcal{W}$= $\{w_{g,n,l} | 1 < w_{g,n,l} \leq MaxWT;\ n = \{1, \ldots, POPU\},\ l = \{1, \ldots, NrLink\}\}$;
Chromosome: $C_{g,n} = \{gene_{g,n,l} | l = \{1, \ldots, NrLink\}\}$;
Population: $C_g = \{C_{g,n} | n = 1, \ldots, POPU\}$;
Parents: $P_g = \{\vec{C}_{g,n} | n = 1, \ldots, POPU\}$;
Off-springs: $O_g = \{\vec{C}_{g,n} | n = 1, \ldots, POPU\}$

Input: a network with nodes $\mathcal{N}$, links $\mathcal{L}$, link capacities $\mathcal{C}$, and flow demands $\mathcal{D}$.

Given: $MaxWT$, G, $POPU$, and fitness function:

$$Fit_{g,n} = f_{Fit}(\vec{C}_{g,n}) = \left(\frac{1}{MaxU_{g,n}}\right)^{p_{g,n}}, where \tag{2.3}$$

$$p_{g,n} = \begin{cases} 0.5 & \text{if } MaxU_{g,n} < MaxU_{g-1,n} \\ 1.5 & \text{if } MaxU_{g,n} > MaxU_{g-1,n} \\ p_{g-1,n} & \text{if } MaxU_{g,n} = MaxU_{g-1,n} \end{cases}$$

in which, in generation g, for the nth chromosome, the Maximum Link Utilization is given by $MaxU_{g,n} = Max_{l\in\ \mathcal{L}}(\mu_l)$.

Output: optimal link weights $\mathcal{W}$= $\{w_l | 1 \leq w_l \leq MaxWT; l \in \mathcal{L}\}$.

Objective: minimizing the maximum link utilization $max_{l\in\ \mathcal{L}}\ (\mu_l)$.

Algorithm: Main Program

Step1: **Input** the network topology and the traffic demands;

Step2: Set the current generation g to 0;

Step3: Randomly initialize the population C_g;

Step4: Calculate fitness values:

Repeat

(a) Route traffic demands $\mathcal{D}$;

(b) Calculate fitness $F_g = \{Fit_{g,n} | n = 1, \ldots, N\}$;

(c) Perform local search;

Until no further improvement to the maximum link utilization;

Step5: Modify selection probabilities $Prob_g = P_{g,n}$;

Step6: Randomly select parents P_g;

Step7: Produce off-springs:

Set $i = 1$;

Repeat

(a) Randomly select a mother M and a father F from P_g;

(b) Perform crossover to M and F to produce off-springs $O_{g,i}$, $O_{g,i+1}$;

(c) Mutate off-springs $O_{g,i}$ and $O_{g,i+1}$;

(d) Set $i = i + 2$;

Until $i = POPU$;

Step8: Replace the population C_g by selecting individuals from C_g and O_g;

Step9: **If** $g < G$ **THEN** set $g = g + 1$; **GOTO Step4**;

Step10: **Output** optimal link weights $w_{g,j}$, by which $\mu_{g,j} = min_{k=0}^{k<POPU}(\mu_{g,k})$,
$0 \leq j \leq POPU$.

Algorithm: Mutation
00: **if** *a random number* $<$ the given mutation probability **then**
01: replace the weight on a randomly selected link with a random value

Algorithm: Replacement
00: copy the best one from the Offspring into the new Population
01: copy the best one from the Population into the new Population
02: **while** not got all elements of the new Population **do**
03: **for** each individual of the Offspring **do**
04: **if** *a random number* $<$ Selection Probability of the individual **then**
05: copy this individual into the new Population
06: **if** got all elements of the new Population **then**
07: **break**
08: **for** each individual of the Population **do**
09 **if** *a random number* $<$ Selection Probability of the individual **then**
10: take this individual into the new Population
11: **if** got all elements of the new Population **then**
12: **break**

Algorithm: FitnessCalculation
00: **for** each individual in the population **do**
01: **call LocalSearch**
02: **if** failed to route demands **then**
03: $Fit_{g,n} = 0$
04: **else**
05: $Fit_{g,n} = (1/MaxU_{g,n})^{P_{g,n}}$

Algorithm: LocalSearch
00: **repeat**
01: **for** each traffic demand **do**
02: **call** Dijkstra to calculate the shortest path
02: route demand along the shortest path
03: search a link l_i holding the maximum utilization MaxU
04: increase the weight of l_i by 2
05: **for** other links **do**
06: **if** a random number < remained bandwidth / $\sum$ remained bandwidth **then**
07: increase the weight by 1
08: **if** *the minimum link weight* > 1
09: **repeat**
10: decrease all weight by 1
11: **until** the minimum link weight = 1
12: **until** no further decreasing on MaxU

2.4 An illustration

Using the USA test network described below, we compared the GA and HybridGA to OSPF routing. While the link weights were set inversely proportional to the link capacity as suggested by Cisco [27] for OSPF routing, the simulation parameters were set to a maximum of 20 chromosomes and 100 generations for the GA and HybridGA algorithms. The USA network depicted by Figure 2.4 has 23 nodes and 76 directional links with some of its links (the thicker lines) having a link capacity of 240 units while others have 120 units of capacity. A total number of 15 origin-destination pairs were considered by the traffic matrix of the network, a number expressing low traffic profile. Note that the performance indexes selected for our illustrations are

- *MLU:* the maximum link utilization expressing the optimality of an algorithm. An algorithm with lower maximum link utilization will be preferred to one with higher maximum link utilization.

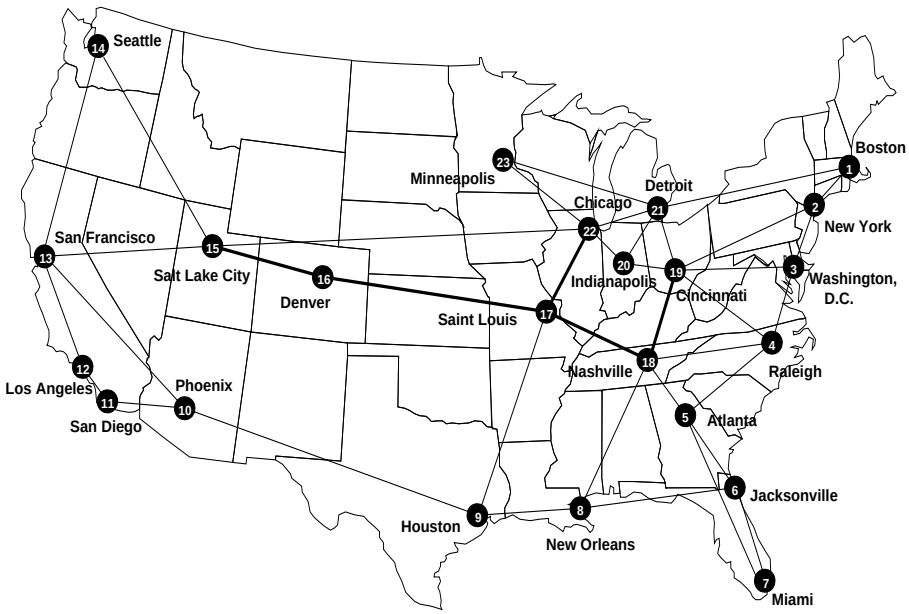


Figure 2.4: Topology of the USA test network

- *ALU:* the average link utilization also expressing the optimality of an algorithm. An algorithm with lower average link utilization will be preferred to one with higher average link utilization.

- *Time:* time spent to complete the evolutionary process. This parameter expresses the scalability of the routing algorithm: a more scalable algorithm will use less time to complete the process.

- *Average path length:* the average length of paths in terms of hop count. This parameter expresses resource consumption since shorter paths will tie up less resources than longer paths.

- *Maximum path length:* the maximum length of paths in terms of hop count. This parameter is also an indication of resource consumption but has less impact on the resource consumption than the average path length.

2.4.1 Comparison on the USA test network

The results depicted by Table 2.1 are average values computed using 20 replications to achieve 95% confidence interval. These results reveal that on most of the performance

Performance	GA	HybridGA	OSPF
MLU	0.33	0.25	0.5
Time	8s	219s	1s
Average path length	4.5	4.4	4.3
Maximum path length	6	6	6

Table 2.1: Performance comparison between GA and HybridGA

indexes, HybridGA outperforms GA and OSPF routing in terms of the routing optimality expressed by the MLU and the resource consumption defined by the path length. However, this relative efficiency is achieved at the price of higher processing times spent likely in the local search procedure.

We conducted another set of experiments using NS simulation [29] to evaluate the throughput achieved by the IGP TE process when routing the traffic offered to the USA network by applying the weights found by GA, HybridGA and using the OSPF weights set inversely proportional to the link capacities. The results depicted by Figure 2.5 reveal that HybridGA achieves the highest throughput and GA the second best throughput while OSPF routing performs worse.

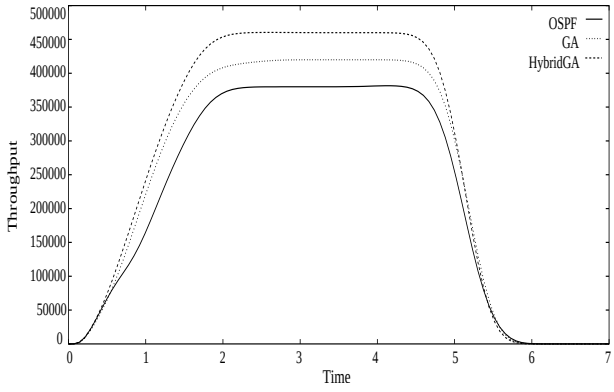


Figure 2.5: Throughput comparison among OSPF, GA, and HybridGA

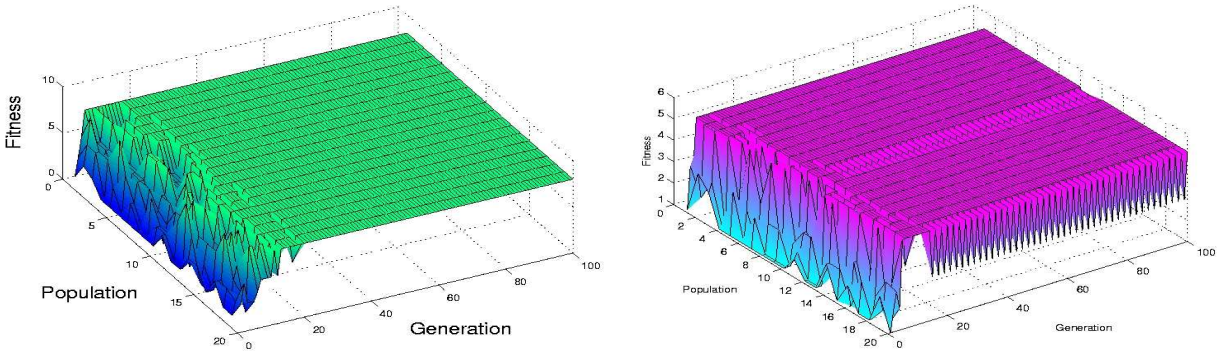
2.4.2 The impact of genetic operations

The convergence to the global optimum is a major issue for the genetic algorithms since good genetic algorithms are expected to quickly converge to the global optimum instead

of being stuck at any local optimum. Looking at Figure 2.6, one can see that while both algorithms converge to their best fitness after the 20th generation, HybridGA find fittest populations compared to GA as expressed by the respective values of 9 for HybridGA and 5 for GA. This relative efficiency of the HybridGA algorithm results from the local search which allows the algorithm to jump out of local optima before the 20th generation. Better convergence could also be achieved by adjusting the probabilities of the genetic operations or by using more complex mutation operations. Such strategies are not discussed in detail in this thesis work.

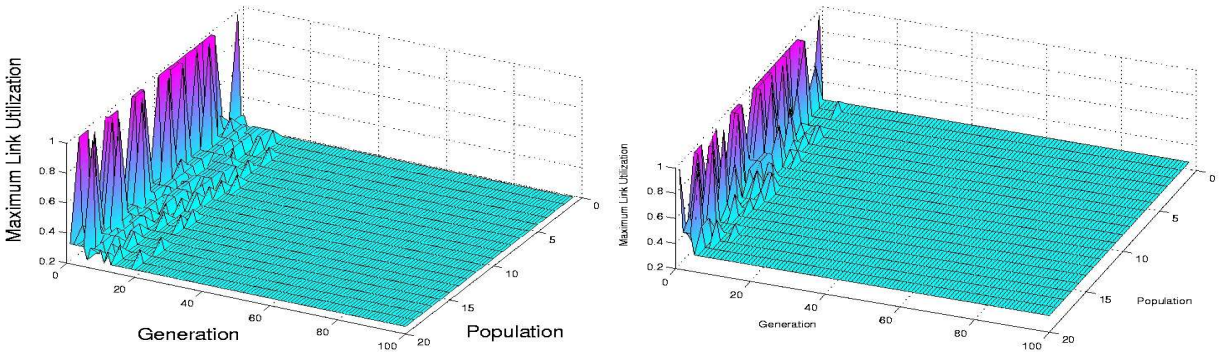
Looking at the maximum utilization achieved by the two algorithms, one can find that HybridGA performed excitingly well on our test network by starting from a randomly initialized population and quickly finding the optimum value. Each further generation following the 20th found the optimum as illustrated by Figure 2.7. Similarly, our GA converged very quickly but to a lower optimum showing that it was stuck at a local optimum as depicted by Figure 2.7. These results reveal that our HybridGA performs better than GA on minimizing the maximum link utilization.

It can also be observed from both figures 2.7 (a) and (b) that while the GA get stuck to its optimum value at almost 0.33 maximum utilization after almost 5 generations, HybridGA fitness function reached a lower value of the maximum utilization at the 20th generation. This also likely results from the local search implemented by HybridGA which through hill-climbing allowed the HybridGA algorithm to converge further towards the global optimum.



(a) HybridGA (b) GA

Figure 2.6: Evolution of the Fitness



(a) HybridGA (b) GA

Figure 2.7: Evolution of the Maximum Link Utilization

The results above suggest that since the optimum value can be reached by an algorithm earlier than the number of generations defined for the genetic evolution, the stopping criterion of GA and HybridGA should be defined based on the improvements achieved by the algorithms over generations rather than being a priory defined as a static routing parameter defining the number of iterations to be executed by the genetic algorithm. This can improve the processing time of the algorithms.

2.5 Conclusion

We have described the GA and HybridGA algorithms for solving the IGP TE problem and illustrated their main differences and genetic features using a USA test network. Based on our illustrative results, we found that:

1. GA solves the IGP TE problem but get stuck on a local optimum value.
2. HybridGA performs better than GA in terms of convergence to the global optimum.
3. HybridGA can be more computationally intensive than GA when deployed with a priory fixed number of generations.
4. To improve on processing time, the stopping criteria of both genetic algorithms should be defined based on the evolution of the objective function values with the generations

rather than being defined a priory as a fixed number used as routing parameter.

Chapter 3

Gene Expression Programming

Gene Expression Programming (GEP) is a new approach to Evolutionary Computation which works by evolving computer programs which are encoded in linear chromosomes of fixed length. GEP was proposed by *Cândida Ferreira* [20] in 1999. It is a development of genetic algorithms (GAs) and genetic programming (GP). GEP borrows from Gene Programming (GP) the diagram representation of the evolutionary entities but uses *expression trees* to represent a genome. As depicted by Figure 3.1, GEP is based on a evolutionary loop similar to GA where new generations are produced by inheriting gene materials from matted parents to off-springs. However, GEP uses more complex genetic operations based on gene expressions. Though initially used to solve optimization problems in different other research fields, GEP can be used to solve the IGP TE problem when a link weight value is expressed as a program (an algebra expression). Contrasting to GA, it has more genetic operations and searches a wider space for the same number of generations. Hence it is expected to converge to the global optimum further than GA.

This chapter presents two implementations of the GEP algorithm: (1) One based on the classic GEP proposed by [20] and its hybrid referred to as HybridGEP which is an improved GEP algorithm using a local search process to improve convergence to the global optimum.

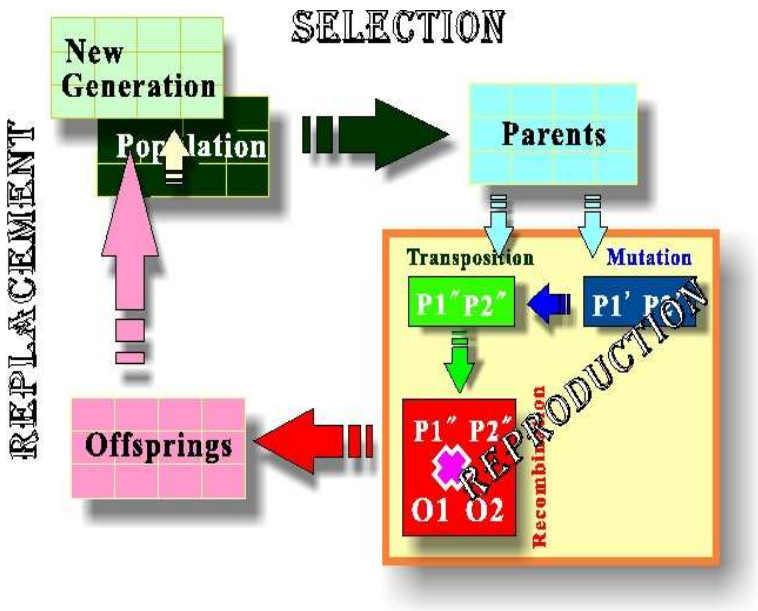


Figure 3.1: Evolution circuit in GEP

3.1 Introduction to GEP

GEP encodes the genetic information into **chromosomes** which are usually composed of more than one gene of equal length. In GEP, a chromosome is expressed by an **Expression Tree (ET)**. A gene is composed of two parts: a **head** and a **tail**. The head contains symbols that represent both **functions** and **terminals**, whereas the tail contains only terminals. For a specific problem, the length of the head h is a constant constrained by the problem, and the length of the tail t is a function of h. Suppose the number of arguments of the function is n, the length of the tail is evaluated by the equation: $t = h(n-1)+1$. So the length of gene expression equals $h + h(n-1) + 1$. An Expression tree (ET) is a representation of a gene expression where the root is a function, each root of the sub-tree is a function too and each leaf is a terminal. A gene expression is the straightforward reading of its ET from the left to the right and from the top to the bottom [20]. The root of the ET is always the starting position of the gene expression. For each problem, the number of genes as well as the length of the head are a priory chosen. Each gene is used to encode a $sub-ET$ and the $sub-ET$s interact with one another to form a more complex multi subunit ET. An illustration of how the link weights are mapped into expression trees is presented in section 3.1.1.

3.1.1 Genetic operators

In GEP, several kind of genetic operators are used to drive the evolutionary process. They are identified as **Selection and Replication**, **Mutation**, **Transposition**, and **Recombination**. The implementation of these operators is problem specific.

Selection and replication achieve the duplication of gene materials from the parents to the children. According to the fitness value and a probability, each of the individuals can be selected to provide its gene materials to the next generation. The fittest individuals have more chance to be selected and replicated.

Mutation changes the genes of the offspring. To avoid the solution to be stuck to a local optimum, some genes have to be changed to generate different gene material in order to widen the search space. The Mutation is applied to a randomly selected position of a gene expression but the structural organization of the mutated chromosome must be maintained.

Transposition reforms the gene expression using the materials inherited from a parent. There are three kinds of transpositions: **Insertion Sequence (IS) Transposition**, **Root Transposition**, and **Gene Transposition**. Transposition is based on probabilities [20].

IS Transposition randomly inserts a part of one gene expression's head to another gene expression's head at a randomly selected position;

Root Transposition randomly takes a part of a gene expression's head with a function at the beginning, and inserts this part into first position;

Gene Transposition randomly takes a gene from the chromosome and inserts this entire gene expression into another randomly selected gene expression's first position.

Recombination exchanges the inherited gene materials between the parents. There are three kinds of Recombination: **One-point recombination**, **Two-point recombination**,

and **Gene recombination**. They are based on probabilities [20].

One-point Recombination exchanges the same part between parents split at a randomly selected position;

Two-point Recombination exchanges the same part between two randomly selected positions between the parents;

Gene Recombination exchanges entire randomly selected genes.

3.1.2 The GEP Algorithm

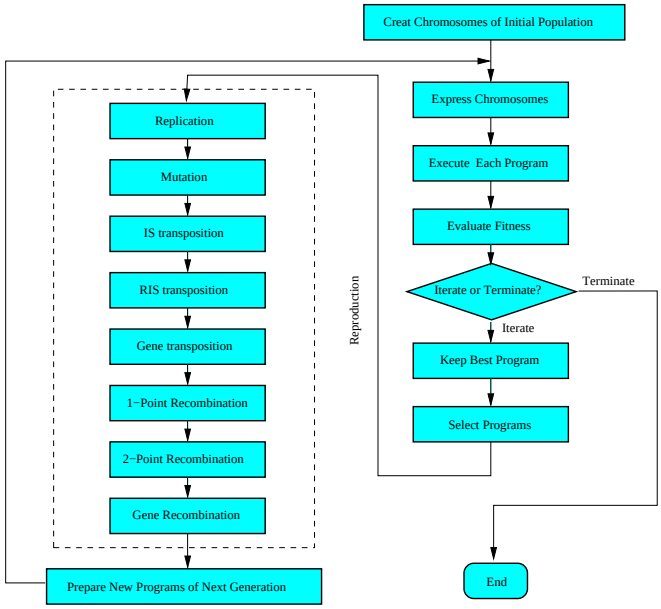


Figure 3.2: Flowchart of a GEP algorithm

Figure 3.2 illustrates the classic GEP algorithm [20]. It is described by the seven steps pseudo-code below

Algorithm: Classic GEP

Notation:

g: index of generation; $\vec{C}_{g,n}$: nth chromosome in gth generation; C_g: population in gth generation; $P_{g,n}$: function expressed by nth chromosome in gth generation; $Fit_{g,n}$: fitness of nth chromosome in gth generation; $Prob_{g,n}$: selection probability of nth chromosome in gth generation; $\vec{O}_{g,n}$: nth offspring(a chromosome) in gth generation.

Step 1: Set the current generation be $g = 0$.

Step 2: Initialize the population $C_g = \{\vec{C}_{g,n} | n = 1, \ldots, N\}$.

Step 3: Fitness calculation and evaluation:

- **(a)** Express chromosomes C_g into programs $P_g = \{P_{g,n} = f_{Prog}(F, T) | n = 1, \ldots, N; F = \{functions\}, T = \{terminals\}\}$.
- **(b)** Execute each program $P_{g,n}$ to get a potential solution $S_{g,n} = f_{out}(\vec{C}_{g,n})$.
- **(c)** Evaluate all fitness $F_g = \{Fit_{g,n} | n = 1, \ldots, N\}$.
- **(d)** Modify selection probabilities $Prob_g = \{Prob_{g,n} = f_{prob}(Fit_{g,n}) | n = 1, \ldots, N\}$.

Step 4: Stop if the stopping condition is met.

Step 5: Use genetic operators to produce off-springs $O_g = \{\vec{O}_{g,n} | n = 1, \ldots, N\}$.

Step 6: Select new individuals of population $C_{g+1,n}$ from C_g and O_g.

Step 7: Set $g = g + 1$. **GOTO Step 3**.

3.2 Application to IGP TE Problem

We consider an IGP TE model where the link weights are mapped into GEP chromosomes and the different genetic operations are used to evolve the evolutionary process to find an optimal set of link weights. This model solves the IGP TE problem described in Chapter 1

using two algorithms: a classic GEP algorithm and its hybrid referred to as HybridGEP. These two algorithms differ only by the use of the local search process in HybridGEP.

3.2.1 Forming chromosome

A chromosome is determined by the number of links, the range of the link weight values, and the encoding of the link weight. We express a link weight as an integer value in the range $[2, M]$ where the upper bound is set to the value $M = 30$. In order to express the link weight, we use three figures: 1, 2, and 5 to form terminals, and use the figure 0 as the sum function instead of "+". The aim is to present a number by summing 1s,2s, and 5s, and use a gene expression to express such a sum function. Because the expression $x+x+x+x+x+x+x$ $(x \in \{1, 2, 5\})$ can represent all integers in the range $[2, 30]$ (e.g. the greatest number being $30 = 5+5+5+5+5+5$, and a number with the longest expression being $28 = 5 + 5 + 5 + 5 + 5 + 2 + 1$), a gene expression consisting of 13 positions (six for the head and seven for the tail) will be long enough for our representation. Therefore a chromosome formed by N genes will have $13 * N$ positions.

For example:
The number 8 can be expressed by 5+2+1. So the gene expression could be 05021 52122212. The gene expression is illustrated in Figure 3.3

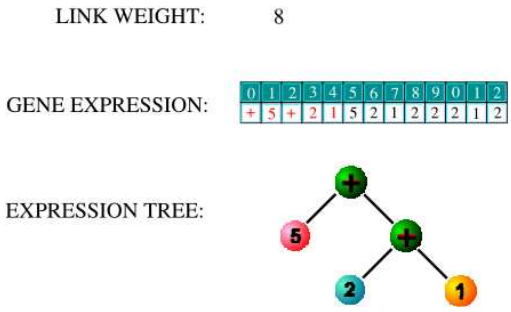


Figure 3.3: Gene expression of link weight 8

Also, the number 23 can be implemented by $5 + 5 + 5 + 5 + 2 + 1$. So the gene expression could be 00550200551 21. The gene expression is illustrated by Figure 3.4

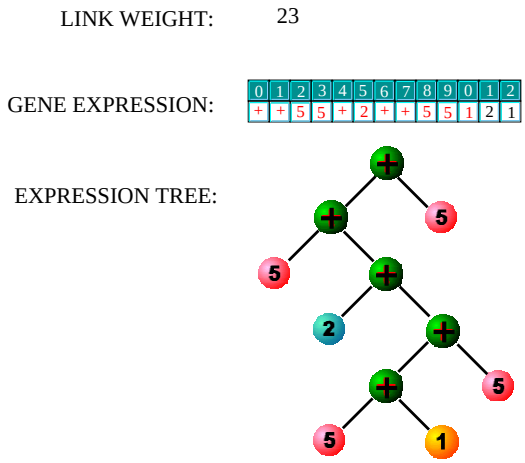


Figure 3.4: Gene expression of link weight 23

Assume a given network has two links with link weight of 8 and 23. A chromosome expressing the link weights is illustrated in Figure 3.5. We use a virtual operator U to group all link weights together to form a chromosome. Note that though cited, the operator U is not involved in any computation in this thesis.

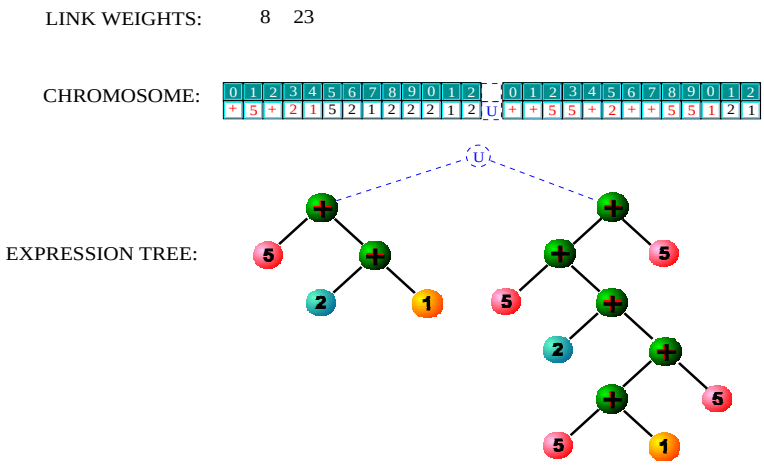


Figure 3.5: Chromosome of link weights 8 23

3.2.2 Fitness Function and Population

The Fitness function used for our GEPs is the same as the one used for the GAs in Section 2.2.3. The Population is also a set of link weights as defined in Chapter 2.

3.3 HybridGEP Algorithm

A flowchart of the HybridGEP algorithm is depicted by Figure 3.6 and a pseudo-code detailing the main program and the different GEP operations/procedures is presented below. Note that since in both GEP and HybridGEP, a link weight is represented by a gene expression though a gene expression can not be directly assigned to a link, a mapping between the expression trees and their link weight values has been added in the evolutionary process to translate the chromosomes into link weights after fitness calculation and before local search using the *Chromosome to Weights* box, and translate the link weights into chromosome after local search and before new fitness calculation using the *Weights to Chromosome* box depicted by Figure 3.6.

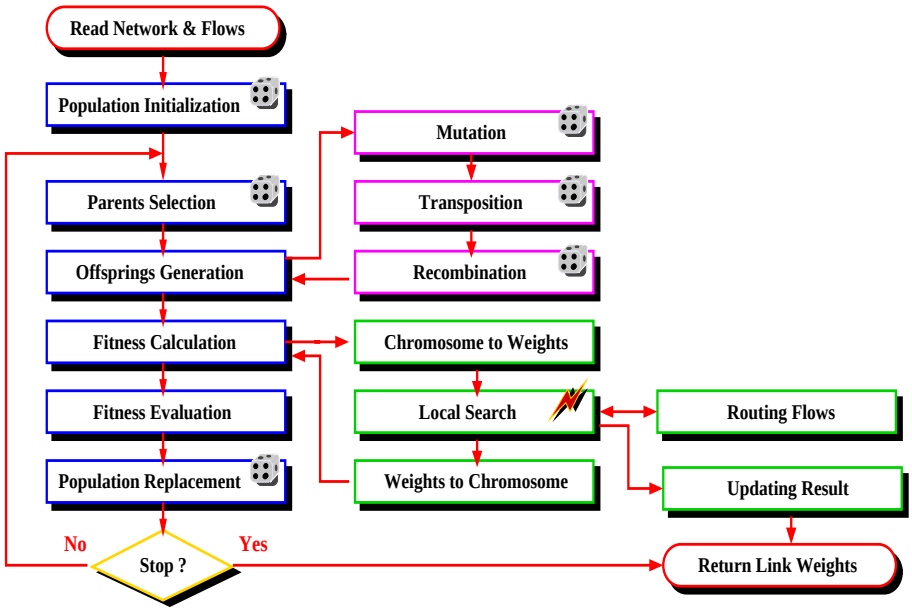


Figure 3.6: Flowchart of HybridGEP

The HybridGEP algorithm consists of a main program and several procedures. The main program implements the basic structure of the GEP algorithm while each procedure implements a specific function.

Purpose: Minimizing Maximum Link Utilization.

Notation:
Number of nodes: $N \in \mathcal{I}$;
Network nodes: $\mathcal{N}$, $\mathcal{N}$= $\{3, \ldots, N\}$;
Number of links: $NrLink$, $1 \leq NrLink \leq N * (N-1)$;
Link: $link_{i,j}$, $i, j \in \mathcal{N}$;
Links: $\mathcal{L}$= $\{l_k | k = \{1, \ldots, NrLink\}\}$;
Link Capacities: $\mathcal{C} = \{c_{i,j} | c_{i,j} \in \mathcal{R};\ i, j \in \mathcal{N}\}$;
Generation: g;
Number of Generation: G;
Number of chromosomes in the Population: $POPU$;
Upper bound of link weight value: $MaxWT$;
Link weights: $\mathcal{W}$= $\{w_{g,n,l} | 1 < w_{g,n,l} \leq MaxWT;\ n = \{1, \ldots, POPU\},\ l = \{1, \ldots, NrLink\}\}$;
Chromosome: $C_{g,n} = \{gene_{g,n,l} | l = \{1, \ldots, NrLink\}\}$;
Population: $C_g = \{C_{g,n} | n = 1, \ldots, POPU\}$;
Parents: $P_g = \{\vec{C}_{g,n} | n = 1, \ldots, POPU\}$;
Off-springs: $O_g = \{\vec{C}_{g,n} | n = 1, \ldots, POPU\}$

Input: A network with Nodes $\mathcal{N}$, Links $\mathcal{L}$, Link Capacities $\mathcal{C}$, and a set of flow demands $\mathcal{D}$.
Given: $MaxWT$, G, $POPU$, and the Fitness function:

$$Fit_{g,n} = f_{Fit}(\vec{C}_{g,n}) = \left(\frac{1}{MaxU_{g,n}}\right)^{p_{g,n}}, where \tag{3.1}$$

$$p_{g,n} = \begin{cases} 0.5 & \text{if } MaxU_{g,n} < MaxU_{g-1,n} \\ 1.5 & \text{if } MaxU_{g,n} > MaxU_{g-1,n} \\ p_{g-1,n} & \text{if } MaxU_{g,n} = MaxU_{g-1,n} \end{cases}$$

in which, the Maximum Link Utilization is given by $MaxU_{g,n} = Max_{l \in \mathcal{L}}(\mu_l)$.

Output: A configuration of optimal link weights $\mathcal{W}= \{w_l | 1 \leq w_l \leq MaxWT; l \in \mathcal{L}\}$.
Objective: Minimize the maximum link utilization $max_{l \in \mathcal{L}}(\mu_l)$.

Algorithm: Main Program

Step1: **Input** network topology and traffic demands;

Step2: Set generation $g \leftarrow 0$;

Step3: Randomly initialize the population C_g;

Step4: Initialize fitness values Fit_g;

Step5: Initialize selection probabilities: $Psel_g$;

Step6: Randomly select parents: P_g;

Step7: Perform Mutation to P_g;

Step8: Perform Transposition to P_g;

Step9: Perform Recombination to P_g to produce off-springs O_g;

Step10: Translate chromosomes $O_g$into link weights W_g;

Step11: Calculate fitness values Fit_g:

Repeat

(a) Route traffic demands $\mathcal{D}$;

(b) Calculate fitness $F_g = \{Fit_{g,n} = f_{Fit}(\vec{C}_{g,n}) | n = 1, \ldots, N\}$;

(c) Local search;

Until no further improvement to the maximum link utilization;

Step12: Modify evolutionary parameters $Psel_g$,etc.

Step13: Translate link weights W_g into chromosomes: C_g;

Step14: Replace the population C_{g+1} by C_g and O_g;

Step15: **IF** $g < G$ **THEN** $g \leftarrow g + 1$; **GOTO Step6**;

Step16: **Output** optimal link weights $w_{g,j}$, by which $\mu_{g,j} = min_{k=0}^{k<POPU}(\mu_{g,k})$, $0 \leq j \leq POPU$.

Algorithm: Randomly Constructing A Chromosome (for Population initialization)

00: **for** each link **do**
01: Set the ROOT to be the function ROOT $\leftarrow$ 0. [a]
02: **for** each position of HEAD **do**
03: randomly select an element from {0,1,2,5} to the position
04: **for** each position of TAIL **do**
05: randomly select an element from {1,2,5} to the position

[a] "0" is the code of function "+"

Algorithm: Parents Selection

00: search the best chromosome of Population
01: select the best chromosome into Parents
02: **while** not got all elements of Parents **do**
03: **for** each chromosome of Population **do**
04: **if** *a random number* $<$ selection probability of the chromosome **then**
05: select the chromosome into Parents

Algorithm: Mutation

00: **for** each chromosome **do**
01: **if** *a random number* $<$ given mutation probability **then**
02: //Mutate a chromosome
03: randomly select a mutation position $Pos \neq ROOT$
04: randomly select a value Val from {0,1,2,5}
05: replace the value at Pos with Val

Algorithm: Transposition

00: **for** each chromosome **do**
01: **if** *a random number* $<$ given transposition probability **then**
02: //Insert Sequence (IS) transposition
03: randomly select a position $Pstr$ in a HEAD
04: randomly select a position $Pdst$ in a HEAD
05: take an IS starting from $Pstr$ with a given length
06: insert the IS at $Pdst$, and within the HEAD, right shift all original values from $Pdst$
07: //Root Insert Sequence (RIS) transposition
08: randomly select a position $Pstr$ of a function
09: random select a ROOT at the position of $Pdst$
10: take a RIS starting from $Pstr$ with a given length
11: insert the RIS at $Pdst$, and right shift all values of original HEAD within the HEAD
12: //Gene transposition
13: randomly select a ROOT position $Pstr$
14: randomly select a ROOT position $Pdst$
15: take the gene start from $Pstr$
16: insert the gene at $Pdst$ and right shift all genes from $Pdst$

Algorithm: Recombination

00: **for** each pair of chromosomes **do**
01: **if** *a random number* $<$ given recombination probability **then**
02: //One-point recombination
03: randomly select a position of P of a chromosome
04: from P exchange the right side part of two chromosomes
05: //Two-point recombination
06: randomly select positions P_1 and P_2 of a chromosome
07: between P_1 and P_2, exchange the sequence of two chromosomes
08: //Gene transposition
09: randomly select ROOT positions P of a chromosome
10: exchange the gene at P of two chromosomes

Algorithm: Replacement

00: select the best from the Offspring into the new Population
01: select the best from the Population into the new Population
02: **while** not got all elements of new Population**do**
03: **for** all chromosomes of the Offspring **do**
04: **if** *a random number* < Selection Probability of the chromosome**then**
05: select the chromosome into the new Population
06: **if** got all elements of the new Population **then**
07: **break**
08: **for** all chromosomes of the Population **do**
09 **if** *a random number* < Selection Probability of the chromosome**then**
10: select the chromosome into the new Population
11: **if** got all elements of the new Population **then**
12: **break**

Algorithm: Translation

00: **for** all links **do**
01: //find the counting sequence of a gene expression
02: $P_{head} \leftarrow 0$
03: $P_{tail} \leftarrow 0$
04: $Step \leftarrow 0$
05: **repeat**
06: $Step \leftarrow 0$
07: **for** each position between P_{head} and P_{tail} **do**
08: **if** the value is 0 [a] **then**
09: $Step \leftarrow Step + 2$
10: $P_{head} \leftarrow P_{tail} + 1$
11: $P_{tail} \leftarrow P_{head} + Step - 1$
12: **until** $Step = 0$
13: $W_{g,n,l} \leftarrow$ sum all values between $ROOT$ and P_{tail}

[a] "0" is the code of function "+"

Algorithm: FitnessCalculation
00: **for** each chromosome **do**
01: **call LocalSearch**
02: **if** failed on routing demands **then**
03: $Fit_{g,n} = 0$
04: **else**
05: $Fit_{g,n} = (1/MaxU_{g,n})^P_{g,n}$

Algorithm: LocalSearch
See the LocalSearch in Chapter 2

Algorithm: **Transformation** Procedure
00: take 5s and remove from weight $w_{g,n,l}$
01: take 2s and remove from weight $w_{g,n,l}$
02: take 1s and remove from weight $w_{g,n,l}$
03: put 0s ahead of 5s, 2s and 1s to form the counting sequence
04: randomly select 5, 2 and 1 to form the rest of the gene
05: shuffle numerals of the HEAD

3.4 An application

Using the same simulation setting as in Chapter 2, we compared the GEP and HybridGEP to OSPF routing using the USA network. While the link weights were set inversely proportional to the link capacity as suggested by Cisco [27] for OSPF routing, the simulation parameters were set to a maximum of 20 chromosomes and 100 generations for the GEP and HybridGEP algorithms like it was done for GA and HybridGA in chapter 2.

3.4.1 Comparison on the USA network

The results depicted by Table 3.1 are average computed over 20 replications to achieve 95% confidence interval. These results reveal that HybridGEP performs slightly better than GEP in terms of resource consumption defined by the path length and better than OSPF

20	GEP	HybridGEP	OSPF
MLU	0.25	0.25	0.5
Time	9s	63s	1s
Average path length	4.7	4.5	4.3
Maximum path length	8	6	6

Table 3.1: Performance comparison between GEP and HybridGEP

routing in terms of optimality expressed by the MLU. However HybridGEP performed well at the price of higher processing times spent likely in the local search procedure. However the trace files not presented in this work showed that HybridGEP reaches its best result in its first generation while GEP only reaches its best result at the 20th generation. As suggested in chapter 2 for the GA and HybridGA algorithms, appropriate stopping rules could therefore be applied to reduce HybridGEP processing time considerably.

We conducted another set of experiments using NS simulation [29] to evaluate the throughput achieved by the IGP TE process when applying the weights found by GEP, HybridGEP and OSPF with link weights set inversely proportional to the link capacities. The results depicted by Figure 3.7 reveal that HybridGEP achieve similar throughput as GEP while OSPF routing performs worse.

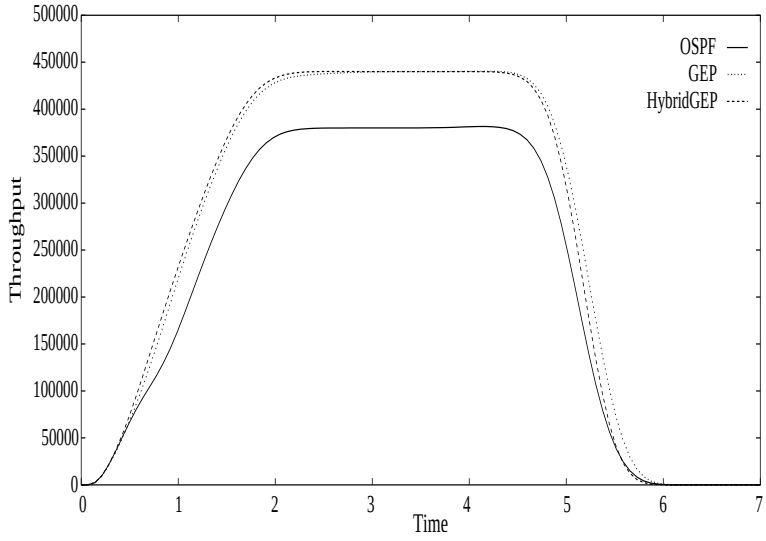
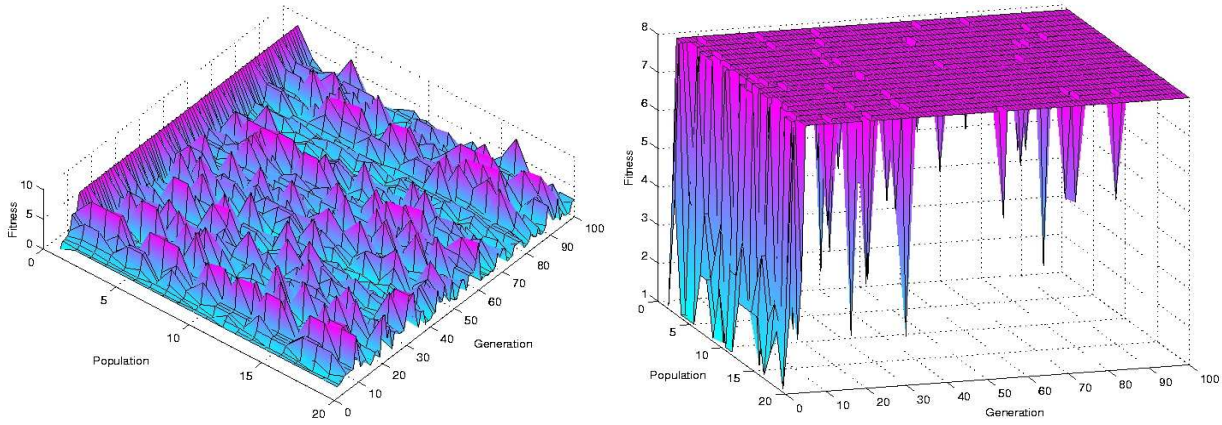


Figure 3.7: Throughput comparison among OSPF, GEP, and HybridGEP

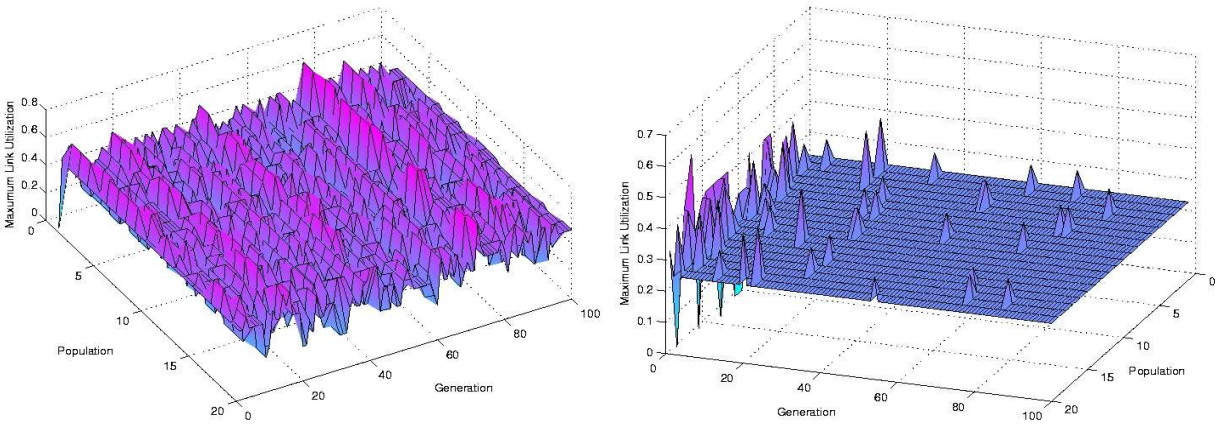
3.4.2 The impact of genetic operations

We conducted another set of experiments to analyze the evolution of the fitness function and the MLU over generations for both GEP and HybridGEP. Figure 3.8 (a) shows that the GEP's fitness frequently fluctuated between higher and lower values. This reveals that GEP does not necessarily lead the fitness to its best value. The reason is that GEP uses many genetic operations which widen the search space and frequently change the link weights. These changes consequently lead to various fitness values. Compared to the GA algorithms which reach a steady state after a number of generations, this is an advantage of using the GEP algorithms since they widen their search space through generations to improve on performance. Figure 3.8 (b) shows that in contrast to GEP, HybridGEP quickly converge to its best fitness. This is likely a result of the the local search process which modifies the link weights in order to evolve toward a global optimum through generations. Looking at the evolution of MLU, we found through Figures 3.9 (a) and 3.9 (b) that similarly to the evolution of the fitness function, the MLU of GEP fluctuate between low and high values While the HybridGEP's MLU quickly converges to its best value as depicted by Figure 3.9 (b).



(a) GEP (b) HybridGEP

Figure 3.8: Fitness evolution



(a) GEP (b) HybridGEP

Figure 3.9: Link Utilization evolution

3.5 Conclusion

This chapter described the GEP and HybridGEP algorithms and presented illustrative results showing how both GEP and HybridGEP can compute paths to minimize the maximum link utilization. Besides their structural differences, GEP performs better than GA. We found that by widening the search space through their genetic operations, the HybridGEP has the potential to converge further than the GA algorithms toward the global optimum. HybridGEP requires more processing time than GEP for the same number of generations. However HybridGEP achieves the best result in less generations so that it is not necessary to complete the same number of generations as GEP and hence reduce its processing time.

Chapter 4

Ant Colony Optimization

Though being almost blind and endowed with a limited amount of memory allowing a random wandering behavior [30, 23], when taken together, real ants can behave cooperatively by using indirect communication through environmental stimuli to achieve complex tasks. These include regulating nest temperature, forming bridges, searching for food, building and protecting their nest, finding the shortest routes to food and exploiting the richest available food source. This form of indirect communication referred to as *stigmergy* is based on two types of changes in the environment:

- **Sematectonic stigmergy** using task-related stimuli in situations where some ants can start performing a task which is used as a stimuli to other ants to contribute to the task. This is for example the case where some ants are digging a hole or building a ball of mud while other are reacting to the stimuli by enlarging the hole or adding more mud to the ball.
- **Sign-based stigmergy** using signal-like stimuli in situations where some ants deposit a volatile hormone referred to as *pheromone* to act as a signal (stimuli) to other ants which will follow the pheromone trail. This leads to an indirect communication based on *pheromone trail following*.

Sign-based stigmergy allows real ants to find the shortest paths from their nest to a food source by exploiting the following key features of the *pheromone trail following* process

- A set of real ants searching for a food source exhibit a random wandering behavior on their search for a food source.
- Upon finding the food source, they return to their nest while laying down pheromone trails on their path.
- Upon finding such a "pheromone marked path", the other ants will exhibit a deterministic behavior following the trail, returning and reinforcing it.
- Over time, the pheromone trail experience an evaporation which reduces its attractive strength.
- As the evaporation is proportional to the time taken by the ants to travel down the path and back again, traversing longer paths will take more time and lead to higher evaporation while shorter paths will be traversed in a shorter time and experience less evaporation.
- As a consequence of pheromone evaporation, the shortest paths will be traversed faster by the ants and show higher pheromone density since pheromone is laid on this path as fast as it can evaporate.

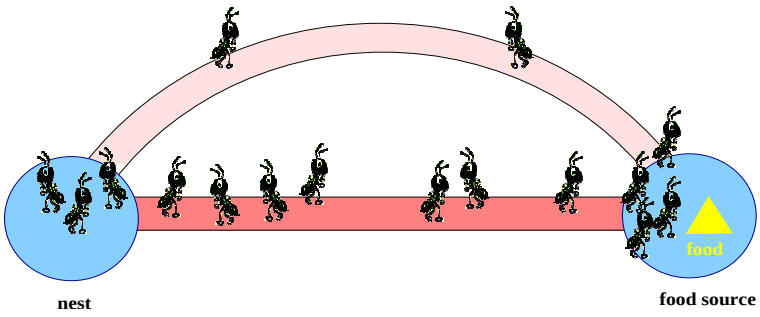


Figure 4.1: Path selection

This process is illustrated by Figure 4.1 where the ants movement between a nest and a food source separated by two bridges is observed: a short red bridge and a longer peach bridge. At the beginning, the ants explore the two bridges with the same probability of reaching the food source and returning to the nest. Eventually with time and evaporation more and more ants will traverse the red shortest bridge and leave stronger pheromone on its path. This is because: (1) the ant routing is based on the *Sign-based "Stigmergy"*

paradigm where more ants follow a bridge with more pheromone (2) the ants following the shortest bridge arrive earlier to the food source or to the nest than others moving on the longer path (3) the red shortest path will carry more ants and have more pheromone on it as suggested by the Sign-based stigmergy process described above and (4) as the shortest bridge is more pheromone marked, most of the ants follow this shortest bridge. Note that a few ants will still explore the longer bridge as an unexplored area when the pheromone has disappeared from it.

This chapter introduces the concept of Ant Colony Optimization (ACO) and its application to the IGP TE problem by presenting and evaluating the performance of two algorithms: (1) ACO which is based on the classic ACO principles but uses a novel mapping between real and artificial ants to find an optimal link weight assignment and (2) HybridACO which introduces hybridization in the ACO algorithm based on the principles previously explained in the previous chapters.

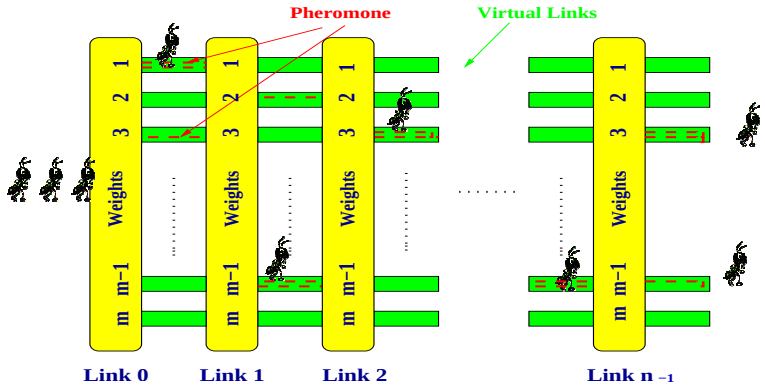


Figure 4.2: The virtual network model

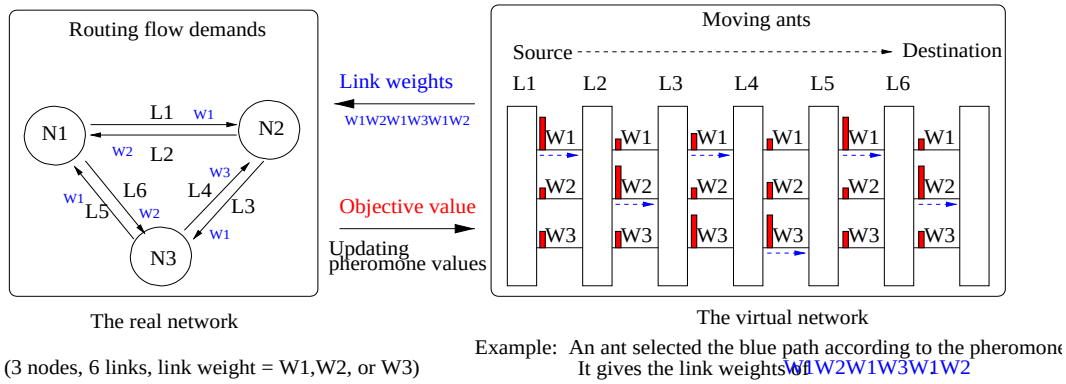


Figure 4.3: The link weight assignment process

4.1 Introduction to ACO

The behavior of real ants using sign-based stigmergy has been applied to real life problems under the Ant Colony Optimization(ACO) [31, 32] denomination. ACO uses a set of cooperative *artificial ants* to solve combinatorial problems by exchanging indirect information via *artificial pheromone* on a graph representing the problem to solve. The **ACO** algorithm is described by M. Dorigo in [33] as follows: A set of computational concurrent and asynchronous agents (a colony of ants) moves through states of the problem corresponding to partial solutions of the problem to solve. They move by applying a stochastic local decision policy based on two parameters, called *trails* and *attractiveness*. By moving, each ant incrementally constructs a solution to the problem. When an ant completes a solution, or during the construction phase, the ant evaluates the solution and modifies the trail value on the components used in its solution. This pheromone information will direct the search of the future ants. The ACO algorithm includes two more mechanisms: *trail evaporation* and, optionally, *daemon actions*. Trail evaporation decreases all trail values over time, in order to avoid unlimited accumulation of trails over some component. Daemon actions can be used to implement centralized actions which cannot be performed by single ants, such as the invocation of a local optimization procedure, or the update of global information to be used to decide whether to bias the search process from a non-local perspective.

4.2 Application to IGP TE Problem

Given a network and flow demands, our ACO algorithm solves the IGP TE problem by letting a given number of artificial ants search a space of link weights to find the optimal link weight assignment minimizing the maximum link utilization (MLU) of a network. This problem is solved by (1) defining a virtual network using artificial ants and pheromones (2) defining an appropriate mapping between the *real network* of routing elements such as links, nodes, link weights and the *virtual network* and 3) using this mapping to derive the optimal link weight assignment which minimize the MLU. As illustrated by Figure 4.2, we consider a *virtual network* model consisting of a grid of nm elements where m different link weight values are associated to each of the n links of the real network. These link weight values referred to as *virtual links* are the m feasible values that a link of the real network

can be assigned to as link weights. As assumed in chapter 2, for OSPF routing, the most used IGP protocols for intra-domain routing, m is usually set to $2^{16} - 1$ to allow the link weight values to be selected in the range $[1, 2^{16} - 1]$. However for scalability reasons, only a subset of this space will be considered in our evaluation.

4.2.1 Mapping between real and virtual networks

The mapping between the real and virtual network is based on the following key features

- While the real network includes real nodes and edges linking the nodes, the virtual network may be considered as a grid of nm elements representing virtual nodes where each virtual node $k \in [1, nm]$ is represented by a pair (ℓ, w_ℓ) where ℓ is a real link and $w_\ell \in [1, m]$ is one of the corresponding link weight values.
- In the *virtual network* depicted by Figure 4.2, each virtual link $w_\ell \in [1, m]$ is associated to the left hand link ℓ and the pheromone dominance on this virtual link reflects the emergence of its associated value as link weight to be used by the real link. The set of virtual links visited by the ant from the first to the last link forms a virtual path.
- While the routing of ants in the virtual network is performed based on the *pheromone trail following* by having the ants move from one virtual node to another following the most pheromone marked virtual links, the routing in the real network aims at finding the least cost paths based on the link weight assignment resulting from the ant routing process. This suggests a feedback loop where virtual and real networks are interacting to achieve link weight assignment as described in section 4.2.2.

4.2.2 The link weight assignment model

Our link weight assignment model is based on a routing process where one by one, according to the pheromone left by former ants and using a probabilistic approach, the ants are routed in the virtual network by **reinforcing** the pheromone left on the virtual links where they passed. The emergence of more pheromone on some virtual links will provide a **feedback**

to the real network through a link weight assignment used to find the least cost paths. As illustrated in Figure 4.3, our IGP TE algorithm follows a feedback loop integrating the ant routing in the virtual network and the traffic routing in the real network to find the best link weight assignment for a given real network. The main steps of the link weight assignment process are

1. **Pheromone initialization.** Initially, the pheromone on virtual links is set to the same value in the virtual network. This will lead the first ant to select randomly any one of the virtual links.
2. **Ant routing.** Each ant is routed based on
   - Pheromone strength by having the most pheromone marked virtual links receive preference.
   - Daemon actions described in section 4.2.5 to avoid solutions which are stuck to local optima.
3. **Link weight selection.** The link weight selection is performed by the virtual network by having the most pheromone marked virtual links to be selected as link weight for the corresponding real link.
4. **Traffic routing.** The traffic routing is performed in the real network by having all traffic requirements expressed by the traffic matrix to be routed based on the link weights selected by the ants in the virtual network to find the least cost paths.
5. **Pheromone update.** The traffic routing is followed by the evaluation of the link utilization for each link of the network and the evaluation of the path length for each least cost path: the maximum link utilization which is also the objective function. The value of this objective function is thereafter used to update the pheromone on the virtual links of the virtual network following the *accumulation, evaporation* process described in section 4.2.4 and the Daemon actions described by section 4.2.5.

Note that as illustrated by Figure 4.3

- The emergence of more pheromone on the virtual links $W1$, $W2$, $W3$, $W3$, $W1$ and $W2$ should lead to the link weight assignment **W1W2W3W3W1W2** assigning the

weights $W1$ to link $L1$, $W2$ to link $L2$, $W3$ to link $L3$, $W3$ to link $L4$, $W1$ to link $L5$, and $W2$ to link $L6$.

- However daemon actions may lead to assigning the link weight value $W1$ to the link $L3$ instead of the value $W3$. This will lead the ant to follow the blue virtual path defined by the virtual links **W1W2W1W3W1W2** instead of the virtual path defined by the pheromone dominance **W1W2W3W3W1W2**.

- The link weight assignment process defines a feedback loop where the routing of traffic in the real network is based on the link weight assignment defined by the ant routing process through pheromone dominance and Daemon actions while the ant path following process uses a pheromone level updating based on the routing objective value computed by the real network.

- The number of ants used in the ant routing process should be selected appropriately to determine the quality of the solution: a higher number of ants may lead to a better solution while a lower number may lead the routing process to get stuck to a local optimum.

- The number of ants should be selected based on the size of the network and the size of the link weight space: using more ants will probably lead to a better solution but for scalability reasons this number should be constrained to an acceptable size.

The relevance of these observations will be assessed later in chapter 6.

Notation:
ℓ_i: a link, where $i \in \{0, \ldots, n-1\}$, and n is the number of links;
$v_{i,j}$: a virtual link, which implies the link weight value j on the link i, where $i \in \{0, \ldots, n-1\}$, $j \in \{1, \ldots, m\}$, m is the upper bound of the link weight value;
B_k: the path found by ant k, where $k \in \{1, \ldots m^n\}$;
$Pheromone_{i,j}$: the pheromone on virtual link $v_{i,j}$.

4.2.3 Calculation of the pheromone

The pheromone left on the virtual links belonging to a path is the inverse of the path length. This allows laying more pheromone on the shortest paths. For example, if the path length is evaluated by $Maximum\ Link\ Utilization(MLU)$, the pheromone may be set to $\alpha/(MLU)^p$ where α is a scaling parameter, and p is a sensitivity parameter. α is used to constrain the pheromone values in a certain range where the probability of randomly selecting a virtual link works well. This is useful when the difference between the objective values found for two paths is very small leading to similar pheromone values causing an ant having hard to select a virtual link having stronger pheromone. p is used to increase the sensitivity of the pheromone to the objective by enlarging the difference of the pheromone values between the virtual links of a real link.

4.2.4 Pheromone accumulation and evaporation

Accumulation and evaporation are used to update the pheromone on all virtual links in order to let some virtual links have stronger pheromone values while the scale of the pheromone is constrained. After an ant has passed a virtual path B_k, the pheromone on all nm virtual links must be updated. There are two actions to update the pheromone: evaporation and accumulation. The accumulation is used to strengthen the pheromone on those virtual links belonging to B_k while the evaporation is used to reduce the pheromone on all virtual links. For each virtual link $v_{i,j}$, the pheromone updating function is defined by:

$$Pheromone_{i,j} = (1-\beta) * \ Pheromone_{i,j} \ + \ \beta * \Delta_{i,j}, \tag{4.1}$$

where $*$ is a multiplicative operator.

$$\Delta_{i,j} = \begin{cases} 0 & \text{if } v_{i,j} \notin B_k \\ 1 - (Length\ of\ B_k) & \text{if } v_{i,j} \in B_k \end{cases}$$

Where β is a parameter balancing accumulation and evaporation while $(1-\beta)* Pheromone_{i,j}$ expresses the evaporation and $\beta*\Delta_{i,j}$ is an expression of the accumulation. Note $\Delta_{i,j}$ won't be negative since the length of B_k is a maximum link utilization value between 0 and 1 leading to $1 - (Length\ of\ B_k) > 0$.

4.2.5 Daemon action

By strengthening the pheromone on the virtual path previously traversed by an ant, the pheromone updating mechanism described above can cause non-least cost virtual paths be more pheromone marked and falsely lead to selecting some less optimal paths as global least cost paths. The daemon action solves this problem by updating the pheromone on all virtual links belonging to the least cost paths. Assuming currently the globally shortest path is S, $S \in \{B_k | k = 1, \ldots, m^n\}$, in Daemon Action, the pheromone on virtual link $v_{i,j}$ is updated by:

$$Pheromone_{i,j} = (1 - \gamma) * Pheromone_{i,j} + \gamma * \Delta_{i,j}, \tag{4.2}$$

where $*$ is a multiplicative operator.

$$\Delta_{i,j} = \begin{cases} 0 & \text{if } v_{i,j} \notin S \\ 1 - (Length\ of\ S)^p & \text{if } v_{i,j} \in S \end{cases}$$

Note: γ is a parameter adjusting the scale of strengthening and $p > 1$ is a scaling factor.

4.2.6 Stopping condition

ACO stops when all ants have found their path, or it may be stopped when the length of the shortest path has not been improved by a certain number of ants. As an example, ACO may stop after routing the 100th ant or when the least cost paths found by the last 10 ants have remained the same.

4.3 The link weight assignment algorithms

We considered two link weight assignment algorithms: one referred to as ACO, based on the classic ACO algorithm while the other referred to as HybridACO includes a hybridization process similar to the one used in previous chapters to improve the global search. Both of these algorithms use the link weight assignment model described in section 4.2.2. A high level description of the ACO algorithm is presented by Figure 4.4 while HybridACO

is depicted by Figure 4.5. Note that ACO is similar to HybridACO except the usage of the local search process.

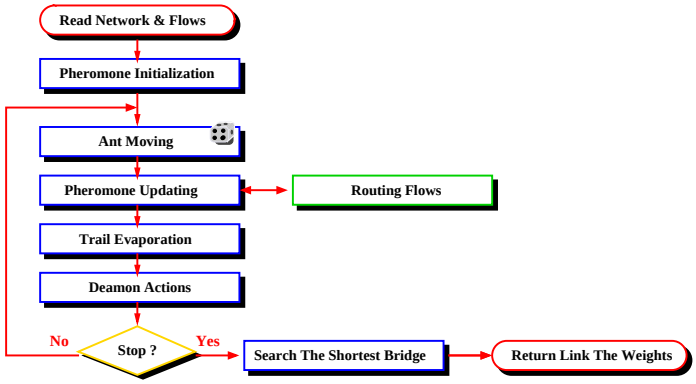


Figure 4.4: Flowchart of ACO

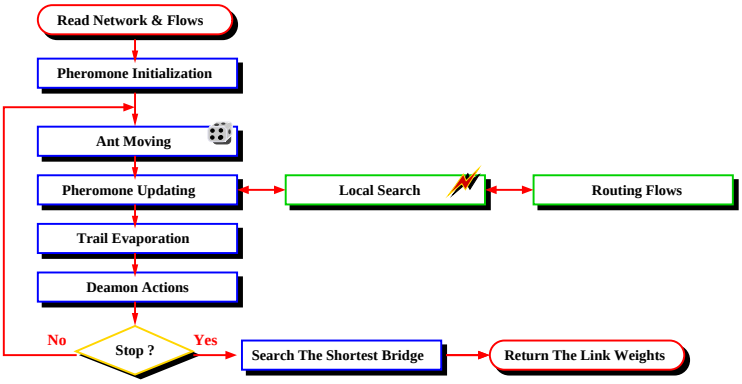


Figure 4.5: Flowchart of HybridACO

A pseudo-code of the main components of the HybridACO algorithm is presented below. It includes the main program, the ant moving algorithm and the Daemon action algorithm.

HybridACO Algorithm: Main Program

00: **Input** network topology and flow demands
01: Initialize pheromone values
02: **For** each ant
03: **For** each link
04: **Call** Moving ant
05: **Call** Updating pheromone
06: **Call** Daemon Action
07: search the shortest path
08: **Return** the shortest path as the optimal link weights

HybridACO Algorithm: Moving ant

Notation P_{jump}: the probability for randomly selecting a virtual link.

00: **Repeat**
01: **For** each virtual link $v_{i,j}$
02: **If** *a random number* $< Pheromone_{i,j}$
03: **Then** select $v_{i,j}$; **Break**
04: **If** *a random number* $< P_{jump}$
05: **Then** randomly select a virtual link
06: **Until** a virtual link has been selected

HybridACO Algorithm: Updating pheromone

00: **Call** Local Search
01: Evaluate the length of path B given by the local search: $l \leftarrow$ MLU of B
02: **For** each link i
03: **For** $j = 1; j <= 30$
04: **If** virtual link $v_{i,j} \in B$
05: **Then** $\Delta_{i,j} \leftarrow 1 - l$
06: **Else** $\Delta_{i,j} \leftarrow 0$
07: Use equation 4.1 to update the pheromone values $Pheromone_{i,j}$.

Algorithm: LocalSearch

See the Local Search algorithm in Chapter 2 for details.

Algorithm: Daemon Action
00: Search the shortest path S
01: Evaluate the length of S: $l \leftarrow$ MLU of S
02: **For** each link i
03: **For** $j = 1; j <= 30$
04: **If** virtual link $v_{i,j} \in S$
05: **Then** $\Delta_{i,j} \leftarrow 1 - l^2$
06: **Else** $\Delta_{i,j} \leftarrow 0$
07: Replace $Pheromone_{i,j}$ by Function 4.2

4.4 An application

We conducted simulation experiments using the same network setting as in previous chapters to illustrate some of the differences between ACO and HybridACO and the impact of some of the ACO operations such as the evaporation and daemon actions on the routing algorithms. In all our experiments, the algorithms were replicated 20 times using different random seeds with the results averaged over the 20 replications to maximize confidence in our results.

4.4.1 Comparison using the USA network

Using 150 ants on the USA network with traffic offered to 15 origin-destination pairs, our first experiments revealed the results depicted by Table 4.1 and Figure 4.6. Table 4.1 compares the performance achieved by ACO and HybridACO based on the following performance indexes:

- *MLU:* the maximum link utilization expressing the optimality of an algorithm. An algorithm with lower maximum link utilization will be preferred to one with higher maximum link utilization.
- *ALU:* the average link utilization also expressing the optimality of an algorithm. An

algorithm with lower average link utilization will be preferred to one with higher average link utilization.

- *Time:* time spent to route the 150 ants. This parameter expresses the scalability of the routing algorithm: a more scalable algorithm will use less time to route the 150 ants.

- *Average path length:* the average length of paths in terms of hop count. This parameter expresses resource consumption since shorter paths will tie up less resources than longer paths.

- *Maximum path length:* the maximum length of paths in terms of hop count. This parameter is also an indication of resource consumption but has less impact on the resource consumption than the average path length.

The results depicted by the figure reveals that on most of the performance indexes, HybridACO outperforms ACO in terms of optimality, resource consumption and scalability of the routing process.

150 ants	**ACO**	**HybridACO**
MLU	0.333	0.250
ALU	0.055	0.052
Time	4s	6s
Average path length	5	4.6
Maximum path length	7	6

Table 4.1: Performance comparison between ACO and HybridACO

Using NS simulation, we conducted another set of experiments to evaluate the throughput achieved by the IGP TE process when applying the weights found by ACO, HybridACO and OSPF with link weights set inversely proportional to the link capacities. The results depicted by Figure 4.6 reveal that HybridACO achieve higher throughput and ACO the second best throughput while OSPF routing performs worse.

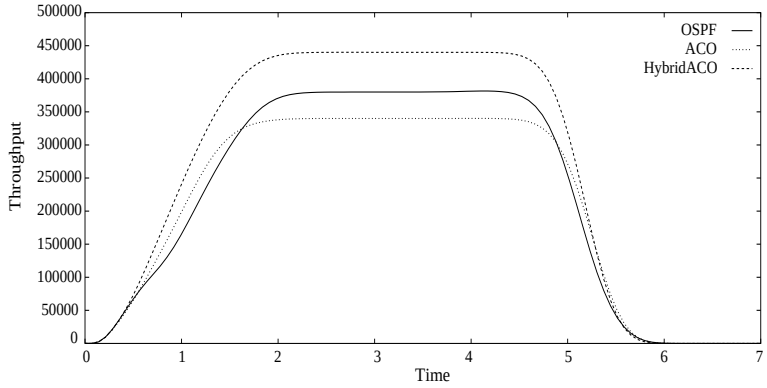
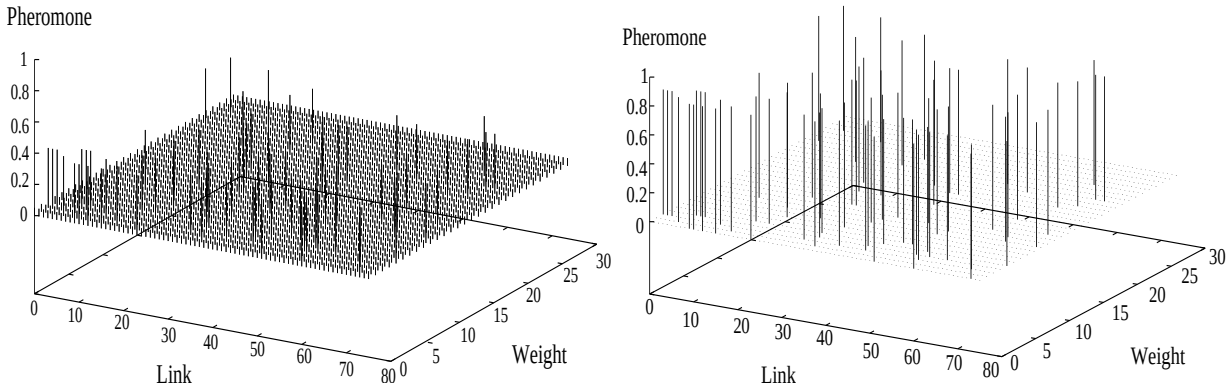


Figure 4.6: Throughput comparison among OSPF, ACO, and HybridACO



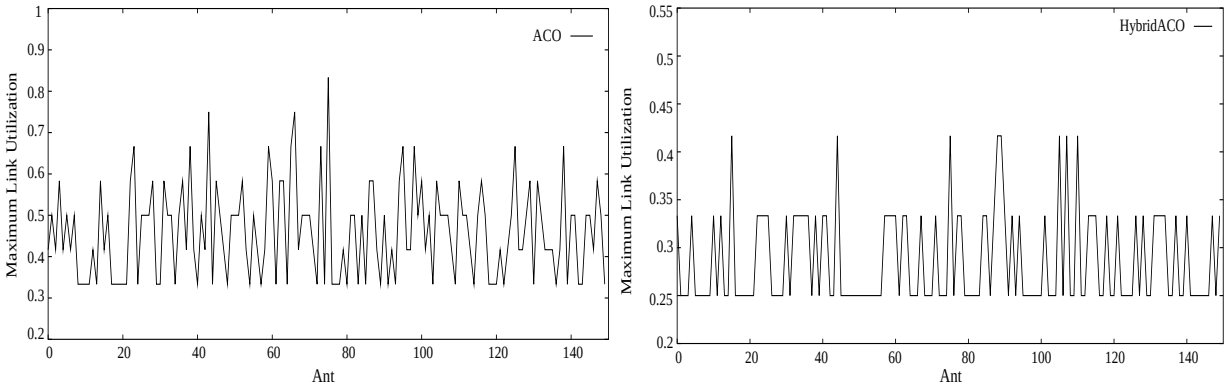
(a) Updated Pheromone (b) After Daemon Action

Figure 4.7: Pheromone updating

4.4.2 The impact of ACO operations

We conducted another set of experiments to evaluate the impact of some of the ACO operations on the performance of the algorithms. These include the pheromone updating operation and the behavior of each of the 150 ants used in our experiments on the MLU achieved achieved by the ACO and HybridACO algorithms. While Figure 4.7 depicts the pheromone updating operations described by equations (4.1) and (4.2), Figure 4.8 reveals the behavior of the ants on the MLU achieved. While Figure 4.7(a) depicts the pheromone remaining on virtual links after after pheromone updating, Figure 4.7(b) shows the pheromone level on virtual links after Daemon action. It can be observed that after pheromone updating, the pheromone is evaporated on most of the virtual links which were not/or rarely selected. Only on those virtual links selected by the last ant or belonging

to some historically least cost will the pheromone remained in higher quantity. From Figure 4.7(b), it can be seen that the Daemon action strengthens the pheromone of the least cost paths. Figure 4.8 compares the MLU values achieved by the ACO and HybridACO algorithms when 150 ants are used to find a link weight assignment in IGP TE. It can be observed from the two figures (a) and (b) that while the ants used by ACO achieve an MLU in the range $[0.35, 0.85]$, the ants used by HybridACO achieve lower MLUs in the range $[0.26, 0.44]$. This relative efficiency of the HybridACO algorithms is likely a result of the hybridization process. Note that these results refer to one of the 20 replications which revealed the worse results: higher maximum utilization.



(a) ACO (b) HybridACO

Figure 4.8: MLU Comparison using 150 ants

4.5 Conclusion

We have presented in this chapter the ACO algorithm and its hybrid referred to as HybridACO. Building upon the ACO principles, we presented a new mapping between real and artificial networks and used this mapping to achieve link weight optimization. Using simulation experiments, we showed that through hybridization, the performance of the ACO algorithm can be improved when routing the traffic to a network to achieve IGP TE by minimizing the maximum link utilization. It should be observed however that, while using the local search process speeds up the convergence of the algorithm to an optimal solution, HybridACO spends extra processing time in the local search but requires less ants to find the same or even better results than ACO.

Chapter 5

Simulated Annealing Algorithms

As described by S. Kirkpatrick [34], Simulated annealing (SA) is a classic nonlinear optimization method which derives from roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure as illustrated by Figure 5.1. The perfect state is that of all atoms lined up on crystal lattice sites with no defects. This is the lowest energy "state" for this set of atoms. For reaching this perfect state, the metal must be heated to very hot to give the atoms energy to move around. Then cool the metal very slowly to gently restrict the range of the motion until everything freezes into the lowest energy configuration. At each temperature, there is a steady state of the substance meaning that energy is locally minimized. Enough time is needed to reach this steady state when the substance is cooled down from a higher temperature.

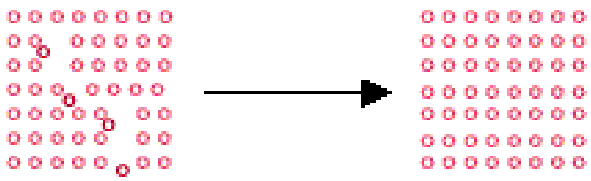


Figure 5.1: Cooling down to freeze the atoms

Comparing to GA, GEP and ACO, SA is simpler and easier to implement but is more time consuming due to failed modifications to the system configuration at each temperature. Obviously, as requested in SA, smoothly modifying the configuration and to slowly cooling down the temperature cause slow convergence of the optimization objective to the optimal

solution. This could be unacceptable in practice when the size of the problem is huge and the objective is not sensitive to the modification of the configuration. However, it is possible to get an acceptable solution by simplifying the classic SA by using hybridization to reduce the time spent on failed configurations.

This chapter presents a simplified version of the SA algorithm refereed to as SA and a hybrid algorithm called HybridSA, both solving the IGP TE problem. Experimental results are provided to discuss the algorithm's performance.

5.1 Introduction to Simulated Annealing

In SA, the potential solution to the optimization problem is considered as the system configuration, and the objective function (i.e. a cost function) corresponds to the energy. A minimum of the objective function corresponds to a steady state of the substance. SA smoothly modifies the configuration at each temperature to reach a steady state meaning a local optimum, and slowly cools down the temperature until the lowest energy state (i.e. lowest cost) is reached. The configuration leading to the lowest energy state is the optimal solution of the given problem.

Notation:
t: the current temperature; t_0: the initial temperature;
C_0 : the initial system configuration;
C: the current system configuration;
C_t: a modified system configuration at temperature t. Note that C is replaced by some C_t according to the acceptance rules which will be given later;
C_{Lowest}: a system configuration corresponding to the lowest energy state;
ΔC: a modification to the system configuration;
E_t: the energy at temperature t;
$E_{Lowest}(t)$: the lowest energy got at all the temperature higher than t;
ΔE: $\Delta E = E_t - E_{Lowest}(t)$, which evaluates the improvement of the energy state. $\Delta E < 0$ means the currently lowest energy is lower than $E_{Lowest}(t)$, (i.e. a better result is found), otherwise the energy state is not improved;
T_{Step}: the temperature cooling factor;
K: the factor for balancing the values between ΔE and t in the calculation of P_{Accept};
P_{Accept}: the probability to accept C_t which leads to a not improved energy state. P_{Accept} is a function of t and E_t, $P_{Accept} = exp(-\Delta E/Kt)$;
r: a random number.

In SA, the initial temperature t_0 must be hot enough so that the system has enough energy to reach the frozen state. Otherwise the probability P_{Accept} of accepting a system configuration C_t will be very small before the system reaches a frozen state. A very low P_{Accept} causes the modification to the system configuration to be hardly accepted and the energy improvement process to be stuck to a given configuration. It is also a requirement that the system configuration be initialized at the beginning to correspond to the initial state of the physical system.

The main part of our SA involves iterations of the cooling process at different temperatures. Repeatedly SA slightly perturb the system to achieve a modification ΔC to the system until the lowest energy $E_{Lowest}(t)$ can not be further decreased at this temperature (i.e. the steady state is reached). Thereafter, the temperature is cooled down and sys-

tem perturbations are performed at this lower temperature in order to reach another lower $E_{Lowest}(t+1)$. SA stops when the $E_{Lowest}(t)$ can not be further decreased (i.e. reaching the frozen state). The system configuration C_{Lowest} is the optimal solution of the given problem.

Each ΔC makes a distinct system configuration C_t. C_t is not always accepted as a new configuration C. There are **acceptance rules** to accept a ΔC and replace the current system configuration C by C_t:

1. To accept a ΔC if $\Delta E < 0$.
2. To accept a ΔC if $\Delta E \geq 0$ but P_{Accept} is greater than a random number r. This second rule helps the objective to jump out from a local optimum.

The way of recognizing the steady state and the frozen state could be various for different ways of modifying the system configuration.

The steady state and frozen state are recognized differently depending on the ways the system configuration is modified. In principle, at each temperature, all feasible configurations should be considered in order to exactly reach the steady state and hence the frozen state. However, this is impossible. According to the second acceptance rule, some configurations are not accepted at each temperature. Especially, for a huge size problem with many configurations, it could be unacceptable to consider each configuration at each temperature for the reason of time consumption. As a result, the C_{Lowest} comes from a part of all feasible configurations, and there is no guarantee that the steady state and the frozen state can always be exactly reached. So it is not necessary to consider all feasible configurations at each temperature though it could lead to a better result.

Normally the simulation stops when the $E_{Lowest}(t)$ has not been improved during a certain number of cooling iterations, and consider the final energy state as the frozen state. However the energy state could be further improved in lower temperature. So we can set a very low temperature as the frozen temperature, and consider the final state as the frozen state.

The SA algorithm is given by the pseudo code below:

Step1: Initialize the system configuration: $C \leftarrow C_0$
Step2: Initialize the temperature: $t \leftarrow t_0$
Step3: Perturb system slightly: to make a C_t based on C
Step4: Compute the energy E_t
Step5: **IF** $\Delta E < 0$

THEN accept this perturbation according to the first acceptance rule: $C \leftarrow C_t$, $C_{Lowest} \leftarrow C_t$, $E_{Lowest}(t) \leftarrow E_t$
ELSE

a) $P_{Accept} \leftarrow exp(-\Delta E/Kt)$
b) IF $r < P_{Accept}$ **THEN** accept this perturbation according to the second acceptance rule: $C \leftarrow C_t$

Step6: IF the energy state is not steady

THEN GOTO Step3

Step7: IF the energy state is not frozen

THEN cool the temperature: $t \leftarrow t * T_{Step}$; **GOTO Step3**
ELSE Stop

5.2 Application to IGP TE Problem

Although the classic SA algorithm can be directly used to solve the IGP TE problem as defined in Chapter 1, we prefer to simplify it in order to deal with huge networks. This is because the process of getting the steady state at a given temperature is a kind of local search leading to high processing overheads and other problems as well. Because, as we discussed in Section 5.1, it is not necessary to consider all configurations at each temperature, we can simply consider only one configuration for each temperature, or we can use an improved local search process to get the steady state. Our SA algorithm is

the first to use such simplification. Our HybridSA algorithm considers the approach of simplifying SA by using our specific local search process as described in Chapter 2. Our SA performs cooling iterations. At a given temperature, the link weight on a randomly selected link is modified (i.e. to make a ΔC). The modified link weights are assigned to the given network. All traffic demands are routed according to this link weights. This modification (i.e. C_t) is accepted if it leads to a better objective value (i.e. E_t) than the historically best objective value (i.e. $E_{Lowest}(t)$). Otherwise, it is accepted if a random number r is less than P_{Accept}. Then SA cools down the temperature and performs the modification again. When the best objective value has not been improved after a certain number of cooling iterations, the SA stops. The link weights leading to the best objective value is the final solution. Figure 5.2 illustrates the flowchart of our SA algorithm. Our

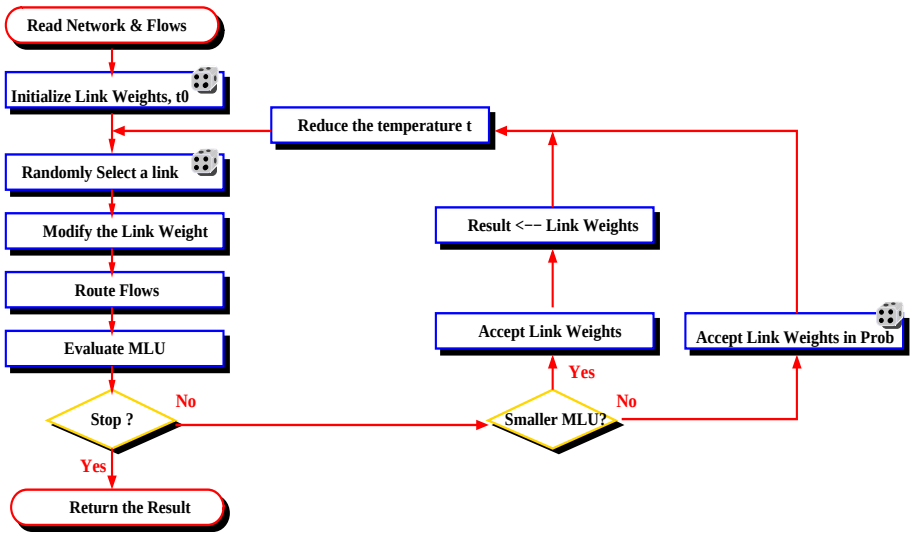


Figure 5.2: Flowchart of SA

HybridSA performs similar cooling iterations. But, at each temperature, after randomly modifying a link weight, our specific local search process is used to search a local optimal configuration. The local search repeatedly increases the link weight on a link having the worst objective value (e.g. the maximum link utilization) until the worst objective value can not be further improved. Our local search finds a local optimum for only one modification. To reduce the time consumption, we accept this local optimum as the steady state at current temperature instead of searching the steady state upon all feasible modifications. Figure 5.3 illustrates the flowchart of our HybridSA algorithm.

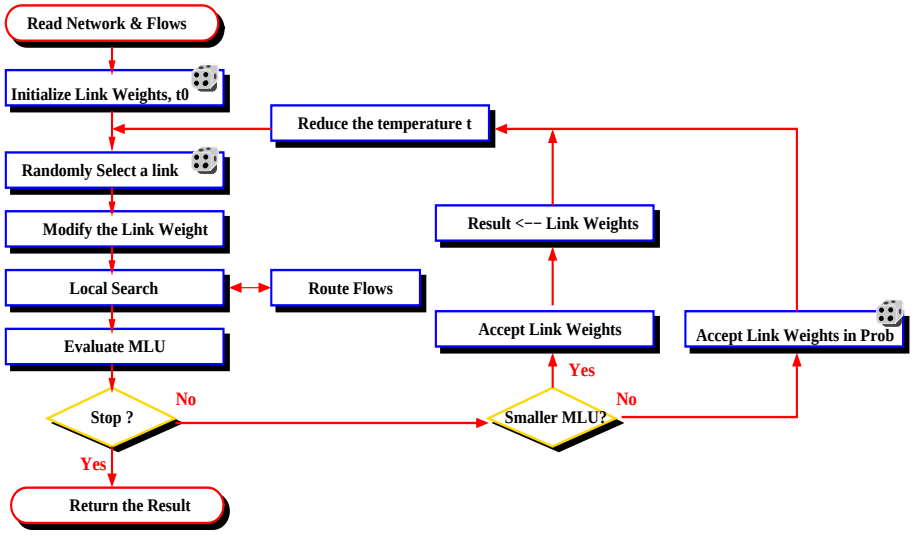


Figure 5.3: Flowchart of HybridSA

5.3 The link weight assignment Algorithms

This section describes SA and its hybrid referred to as HybridSA using the two pseudo-codes presented below.

The SA Algorithm

00: **Input** network topology and flow demands
01: $t \leftarrow t_0$
02: $l \leftarrow 0$ //iteration counter for stopping
03: $E_{Lowest}(t) \leftarrow 1$ //the upper bound of MLU
04: $C \leftarrow C_0$
05: **while** $l <$ *given number of iterations* **do**
06: modify the weight on a randomly selected link
07: route all traffic demands
08: search the MLU
09: $P_{Accept} \leftarrow e^{-\alpha(MLU - E_{Lowest}(t))/t}$ // α is a balancing factor
10: **if** $MLU < E_{Lowest}(t)$
11: **then** //accept the modification
13: $C_{Lowest} \leftarrow C_t$ //keep the optimal link weights
14: $C \leftarrow C_t$ //accept the modification
15: $E_{Lowest}(t) \leftarrow MLU$
16: $l \leftarrow 0$
17: **else**
18: **if** *a random number* $< P_{Accept}$ //accept the modification in a probability
19: **then** //accept the modification
20: $C \leftarrow C_t$ //accept the modification
21: **else** //restore the link weight
22: $l \leftarrow l + 1$
23: $t \leftarrow t * T_{Step}$
24: **Return** C_{Lowest} as the optimal link weights

The HybridSA Algorithm

00: **Input** network topology and flow demands
01: $t \leftarrow t_0$
02: $l \leftarrow 0$ //iteration counter for stopping
03: $E_{Lowest}(t) \leftarrow 1$ //the upper bound of MLU
04: $C \leftarrow C_0$
05: **while** $l <$ *given number of loop* **do**
06: modify the weight on a randomly selected link
//Begin of local search process
07: $MinU \leftarrow 1$ //$MinU$ keeps the currently lowest MLU
08: **repeat**
09: **for** each flow demand **do**
10: **call** Dijkstra // to calculate the shortest path
11: route demand along the shortest path
12: search a link l_i having the MLU;
13: increase the link weight on l_i: W_i++
14: **if** $MLU < MinU$
15: **then** $MinU \leftarrow MLU$
16: **else** restore the weight on l_i: W_i--
17: **until** no further decreasing to $MinU$
//End of local search process
18: $P_{Accept} \leftarrow e^{-\alpha(MinU - E_{Lowest}(t))/t}$
19: **if** $MinU < E_{Lowest}(t)$
20: **then** //accept the modification
21: $C_{Lowest} \leftarrow C_t$ //keep the link weights
22: $C \leftarrow C_t$ //accept the modification
23: $E_{Lowest}(t) \leftarrow MinU$
24: $l \leftarrow 0$
25: **else**
26: **if** *a random number* $< P_{Accept}$ //accept the modification in a probability
27: **then** //accept the modification
28: $C \leftarrow C_t$ //accept the modification
29: **else** //restore the link weight
30: $l \leftarrow l + 1$
31: $t \leftarrow t * T_{Step}$
32: **Return** C_{Lowest} as the optimal link weights

The Initial Configuration

Although the initial configuration can be randomly generated, we prefer to set each link weight to be one, so that we will have the most potential modifications, and the least modified link weights contrasting to OSPF.

The Link Weight Modification

There are options for modifying the link weight. We can increase or decrease a link weight with a certain number. Or we can set it to be a random number. However, in IGP TE problem, on a specific link, to modify the link weight could change the objective value for nothing. As a result, many of such useless configurations waste tremendous cooling iterations. To avoid such situation, our algorithms only deals with the modification leading to values differing to the objective value in the last iteration. What we do is to continually increase the link weight by one until the objective value is changed, or to ignore the modification if the link weight reaches the given upper bound before reaching a different objective value.

The Cost Function

Given the utilization of link i by:

$$Utl_i \;=\; \frac{Used\ Bandwith_i}{Link\ Capacity_i} \tag{5.1}$$

The cost function is given by the objective function:

$$MLU \;=\; \max_{i=1,\ldots,n}(Utl_i) \tag{5.2}$$

where n is the number of links; MLU takes the value of the maximum utilization among all links.

The Acceptance Rules

1. To accept the modified link weights if the assignment of link weights leads to a lower MLU than the lowest MLU at higher temperatures.

2. To accept the modified link weights with a probability:

$$P_{Accept} = e^{-\frac{\alpha(MLU - E_{Lowest}(t))}{t}} \tag{5.3}$$

where α is a constant for balancing the values between t and the difference of MLU and $E_{Lowest}(t)$.

The Cooling Schedule and Stopping Criterion

The value of the initial temperature t_0 affects P_{Accept}. t_0 must be set properly to ensure that, in each iteration, P_{Accept} takes a reasonable value. In our experiments, we set t_0 to be hot enough, e.g. 100 or 1000 units, in order to freeze the objective to an expected solution before the temperature is too low leading to the fact that no configuration will be accepted. In each cooling iteration, we slowly cool down the temperature by multiplying the T_{Step} factor of 0.995. The cooling process stops when the objective has not been changing during a given number of iterations.

5.4 An application

We conducted experiments to compare SA and HybridSA and assess the impact of different annealing operations on these algorithms. Using the same simulation environment as used for HybridGA described in Chapter 2, we collected the results in Table 5.1 and Figure 5.4. These results are averages computed over 20 replications to achieve 95% confidence interval.

$Loop = 100, T_o = 1000, \alpha = 0.25$	**SA**	**HybridSA**
MLU	0.250	0.250
ALU	0.049	0.049
Time	9s	1s
Average path length	4.4	4.4
Maximum path length	6	6

Table 5.1: Performance comparison

5.4.1 Comparison on USA network

The results depicted by Table 5.1 reveal that SA and HybridSA achieve the same performance in terms of optimality and resource consumption expressed by the quality of the paths found. In contrast to genetic algorithms presented in the previous chapters where hybridization increases the processing time, combining hybridization with the annealing process results in lower processing times.

Using NS simulation, we conducted another set of experiments to evaluate the throughput achieved by the IGP TE process when applying the weights found by SA, HybridSA and OSPF with link weights set inversely proportional to the link capacities. The results depicted by Figure 5.4 reveal that HybridSA and SA achieve the same and higher throughput compared to OSPF routing.

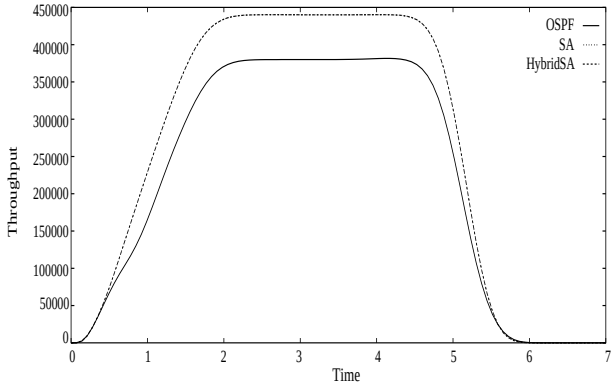


Figure 5.4: Throughput comparison among OSPF,SA,and HybridSA

5.4.2 The impact of annealing operations

The convergence of the algorithm

As seen in Figure 5.5, when using the USA network, SA converged to its optimum value in more than 80 cooling iterations while HybridSA used less than 20 iterations. This reveals that HybridSA converges faster than SA. However there was no guarantee that the objective will always be frozen at the global optimum. This is because, for our SA and HybridSA, the set of accepted link weights is a subset of the potential solution space. This subset is constrained by the algorithm itself and the number of iterations. If the globally optimal solution is not included in this subset, the best objective value is not the global optimum.

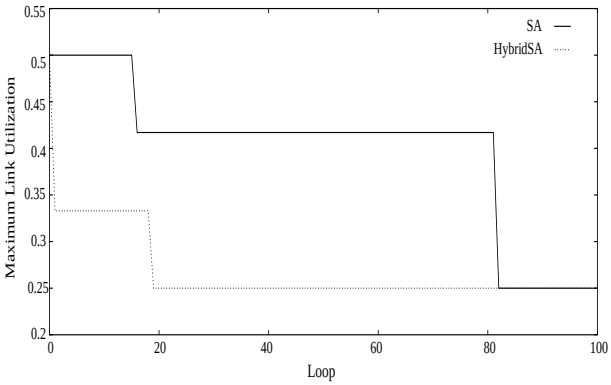


Figure 5.5: Comparison of Maximum Link Utilization between SA and HybridSA

The initial temperature and stopping criterion

The initial temperature and the stopping criterion affect the number of cooling iterations and the acceptance of modifications, hence the final result. For a specific problem, SA and HybridSA can use the same initial temperature and the same stopping criterion, or they can be different. The initial temperature must be hot enough so that in each cooling iteration P_{Accept} takes a value in a reasonable scale. SAs should stop when the frozen state is reached. We can accept an energy state as the frozen state if the objective value has not been changed during a certain number of cooling iterations, or the energy value has

remained the same during a certain number of cooling iterations. For a huge problem, the cooling process could use tremendous number of iterations which is unacceptable in practice. In this situation, the stopping criterion has to be defined as achieving a certain number of cooling iterations or to reach a given lower temperature. Figure 5.6 presents the temperature curves and the points where SA and HybridSA firstly find their best solution. The point on the curve is the upper bound of the frozen temperature which implies the least number of cooling iterations. In this figure, the initial temperature is 100 units. While SA reaches the best solution in its 82th iteration at a temperature of about 67 units, HybridSA reaches the best solution in its 19th iteration at a temperature of about 91 units. The initial temperature is fine for both SA and HybridSA. HybridSA achieves higher upper bound of the frozen temperature than SA meaning that, for the same stopping criterion, HybridSA needs less number of iterations than SA. Note that the best solution given by HybridSA may be different to what given by SA.

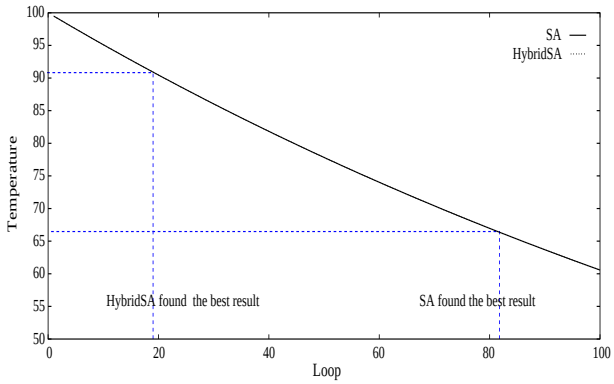
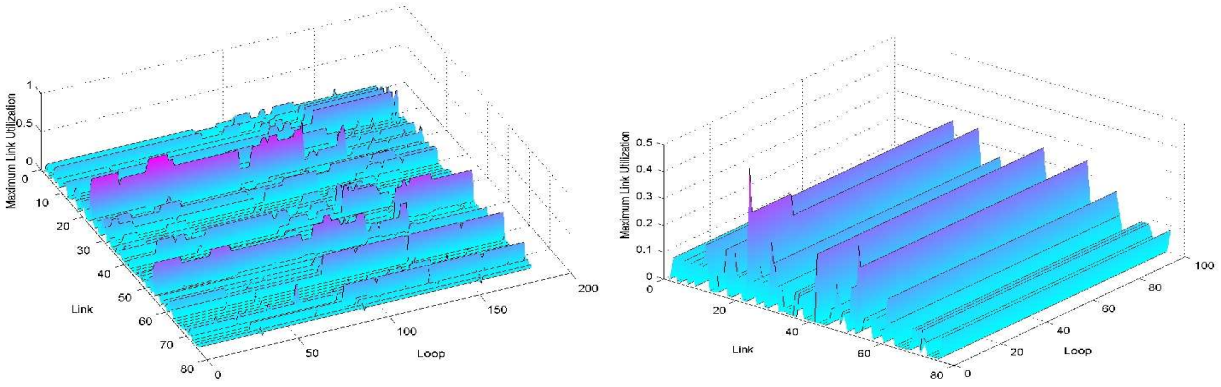


Figure 5.6: Comparison of the Temperature Curve between SA and HybridSA

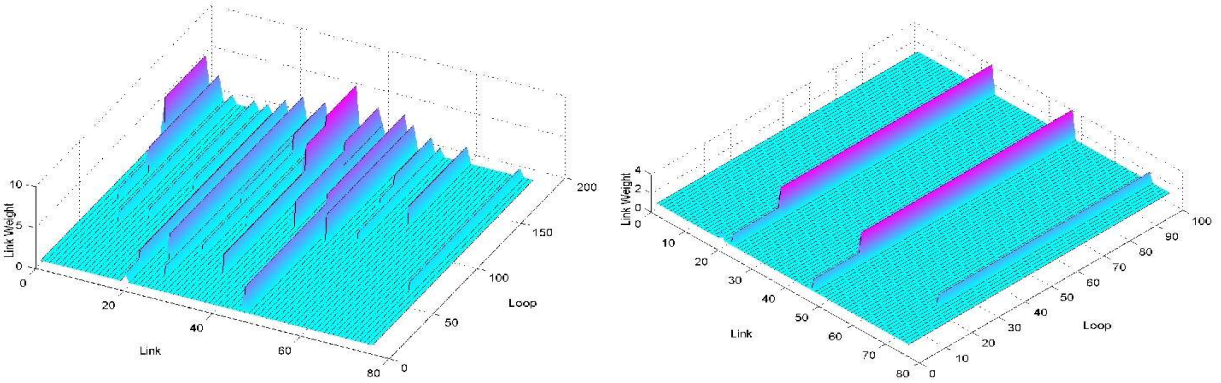
The acceptance of modification and objective

Figure 5.7 (a) shows how the link utilization values were changed by SA during 200 cooling iterations. The figure reveals that the MLU was reduced on different links. Figure 5.7 (b) shows how the link utilization values were changed by HybridSA during 100 cooling iterations. It is clear that, contrasting to SA, HybridSA reduced the MLU down to the best value very early then kept it during the rest of the iterations. It kept steady because there was not any accepted modification.



(a) SA (b) HybridSA

Figure 5.7: Modification to link utilization



(a) SA (b) HybridSA

Figure 5.8: Modification to link weights

We conducted another experiment to compare the link weights found by both SA and HybridSA. The results showed that SA and HybridSA achieved the same performance using different link weight assignments as revealed by the modification of link weights depicted by Figure 5.8. These figures reveal that SA changed 14 link weights as shown by Figure 5.8 (a) while HybridSA changed only 3 link weights as depicted by Figure 5.8 (b). HybridSA changed less link weights because it ignored some modifications to the link weights.

5.5 Conclusion

SA and HybridSA were described and applied to the IGP TE problem in this chapter. While each of these algorithms achieves lower MLU than OSPF, our illustrative results reveal that HybridSA converges faster and produces better results than SA.

Chapter 6

Algorithm Comparison

We have presented in the previous chapters four global search algorithms and applied these algorithms to the IGP TE problem using a classic and a htbrid version built around a local search process. In these chapters, the comparisons provided only an illustrative view of the performance of our algorithms since they were constrained by the test networks using light traffic profiles. By comparing the classic algorithms with their hybrid improvements, we found that each hybrid algorithm performs better than the classic algorithm. Therefore, in this chapter, we will be only interested in the difference among the hybrid algorithms and OSPF routing on networks with higher traffic profiles. We also analyze the impact of different design parameters on the performance of the algorithms.

Our optimization objective consists like in previous chapters of optimizing the Maximum Link Utilization (MLU). The main performance parameters considered in our experiments are the MLU and the Average Link Utilization (ALU) given by:

$$ALU = \frac{\sum_{i=1}^{n} Used\ Bandwidth_i}{\sum_{i=1}^{n} Capacity_i} \tag{6.1}$$

6.1 Introduction to the experiments

In our experiments, various tests were completed for different purposes. They were performed using the following hardwares, softwares and test networks.

Hardware Configuration

* CPU:$Intel$ 750
* Memory: 256 MB
* Hard Disk: 15 GB (1GB $free$)

Software Configuration

* OS:$Mandrake\ Linux\ v10.0$
* Simulator: $ns\ v2.0$

Test Networks

We considered four test networks representing (1) USA network; (2) Europe network; (3) African network and (4) China network. The main features of these test networks are as follows:

USA (23 nodes; 76 links. See Figure 6.1); $Europe$ (30 nodes; 90 links. See Figure 6.2); $Africa$ (31 nodes; 128 links); $China$ (48 nodes; 145 links. See Figure 6.3).

Flow Demands

The traffic demands are given in following files:
Usa (15 O-D pairs); $Usa-all-1$ (506 O-D pairs); $Usa-d1$ (50 O-D pairs); $Usa-d2$ (100 O-D pairs); $Usa-d3$ (150 O-D pairs); $Usa-d4$ (200 O-D pairs); $Usa-d5$ (250 O-D

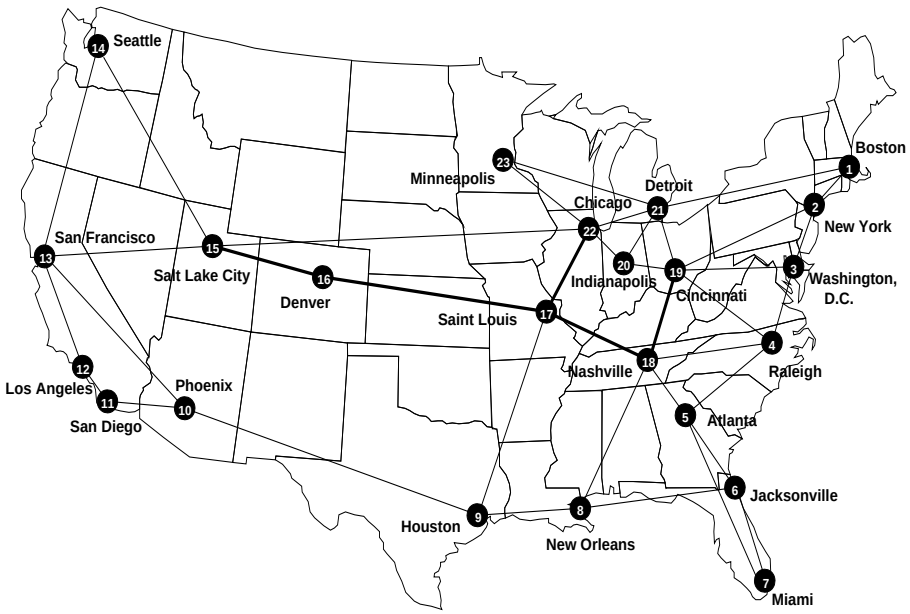


Figure 6.1: Topology of the USA test network

pairs); $Usa - d6$ (300 O-D pairs); $Usa - d7$ (350 O-D pairs); $Usa - d8$ (400 O-D pairs); $Usa - d9$ (450 O-D pairs); $Usa - d10$ (506 O-D pairs); $Euro$ (20 O-D pairs); $Euro - all - 1$ (870 O-D pairs); $Africa - all - 1$ (930 O-D pairs); $China - all - 1$ (2256 O-D pairs). Note that under light traffic profiles, a lower number of O-D pairs are routing the traffic offered to the network while a higher number is an expression of heavy traffic profile. In our experiments, the $Usa - all - 1$, $Euro - all - 1$, $Africa - all - 1$ and $China - all - 1$ are representatives of networks with higher traffic profile.

Executable Programs

Our algorithms were implemented in $C++$ and compiled to following executable files: *gep*; *ga*; *aco*; *sa*. These files can be obtained on demand.

Figure 6.2: Topology of the Europe test network

6.2 Defining confidence intervals for the experiments

We conducted a first set of experiments to evaluate the number of replications which are required to maximize the confidence in our results. The experimental results depicted by the figures 6.4, 6.5, 6.6, and 6.7 revealed that a maximum of 10 replications was sufficient to achieve 95% confidence interval when routing the traffic offered to the Euro network under heavy traffic profile. Similar results revealed that a maximum of 20 replications could achieve 95% confidence in the results produced by the USA, Europe, Africa and China networks under heavy traffic profile. These results have not been reported in this chapter. A comparison of the different average MLU values of algorithms computed at 95% confidence interval is illustrated by Figure 6.8. This figure reveals that with higher confidence (higher number of replications), HybridACO and HybridGEP achieve a better

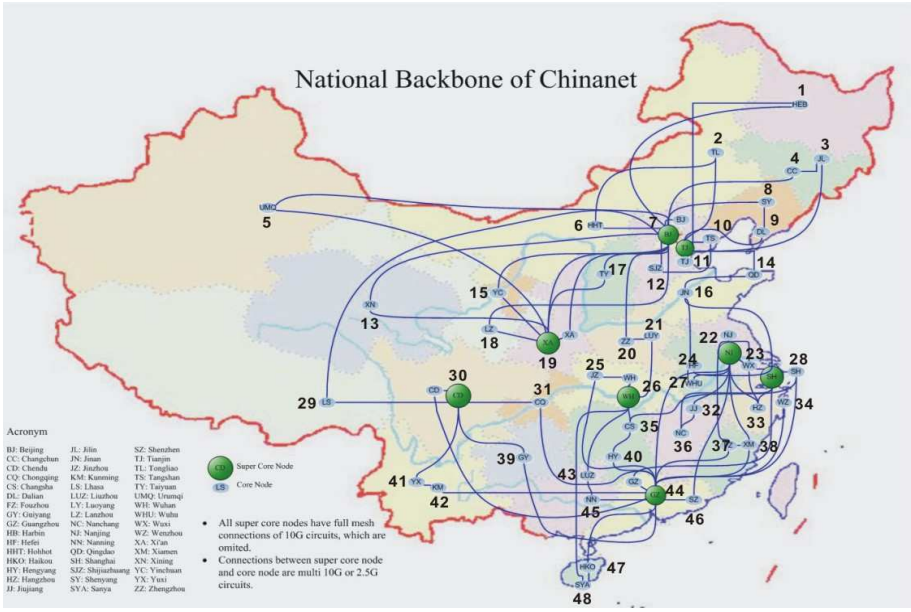


Figure 6.3: Topology of the China test network

MLU compared to the other algorithms. We refer the reader to [35] for more details on the theory of cumulative mean and the confidence interval.

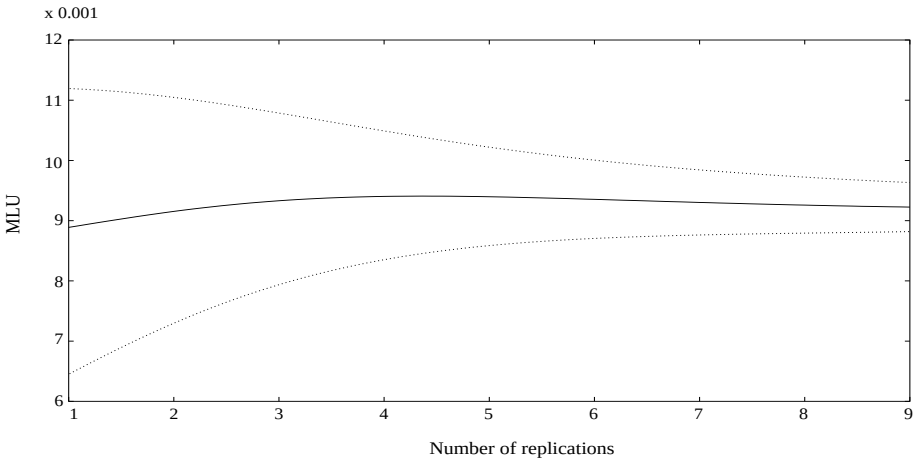


Figure 6.4: Confidence interval of HybridGA on Europe network

USA	**MLU**	**ALU**	**Parameters**	**Used Time (s)**
HybridGA	0.280	0.1168	Popu=60,G=10	2290
HybridGEP	0.280	0.1155	Popu=60, G=20	509
HybridACO	0.290	0.1182	150 ants	201
HybridSA	0.350	0.1128	T0=1000, loop=100, $\alpha = 100$	9
OSPF	0.420	0.1128		< 1
Euro	**MLU**	**ALU**	**Parameters**	**Used Time (s)**
HybridGA	0.017	0.0080	Popu=60,G=10	12201
HybridGEP	0.021	0.0089	Popu=60,G=20	2563
HybridACO	0.013	0.0088	150 ants	934
HybridSA	0.018	0.0086	T0=1000, loop=100, $\alpha = 100$	39
OSPF	0.100	0.0086		< 1
Africa	**MLU**	**ALU**	**Parameters**	**Used Time (s)**
HybridGA	0.203	0.0689	Popu=60,G=10	9242
HybridGEP	0.188	0.0688	Popu=60,G=20	2209
HybridACO	0.180	0.0682	150 ants	872
HybridSA	0.205	0.0629	T0=1000, loop=100, $\alpha = 100$	49
OSPF	0.260	0.0629		1
China	**MLU**	**ALU**	**Parameters**	**Used Time(s)**
HybridGA	0.402	0.1297	Popu=60,G=10	60477
HybridGEP	0.392	0.1264	Popu=60,G=20	20398
HybridACO	0.386	0.1293	150 ants	8912
HybridSA	0.410	0.1207	T0=1000, loop=100, $\alpha = 100$	458
OSPF	0.460	0.1207		2

Table 6.1: Algorithm comparison under heavy traffic profile

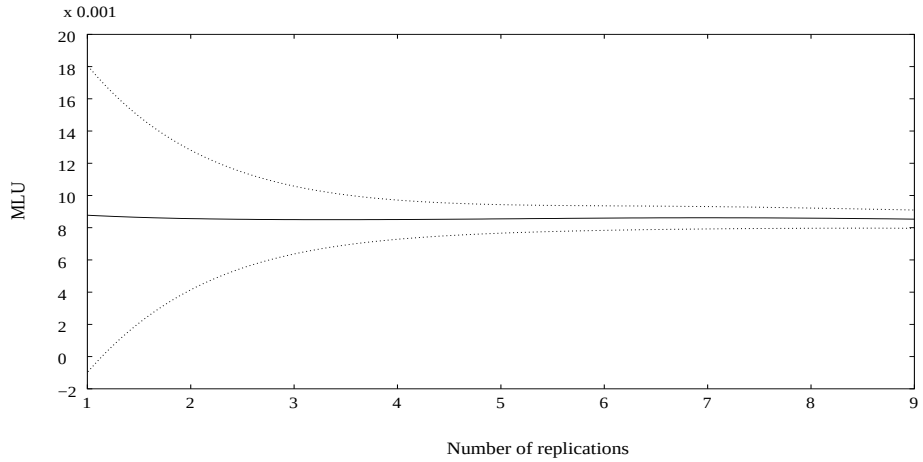


Figure 6.5: Confidence interval of HybridGEP on Europe network

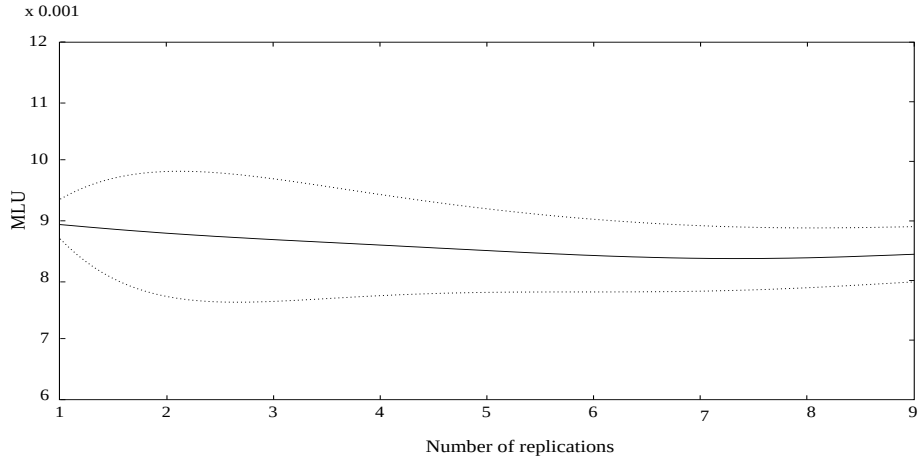


Figure 6.6: Confidence interval of HybridACO on Europe network

6.3 Comparing the different algorithms

We conducted another set of simulation experiments to compare the behavior of our algorithms under heavy traffic profile on the different test networks. The performance values considered in this section are mean values computed at 95% confidence. The results depicted by Table 6.1 reveal a common performance pattern where

- In most experiments, HybridACO performs better than the other algorithms in terms of MLU and ALU.
- HybridGEP achieves in general the second best performance.

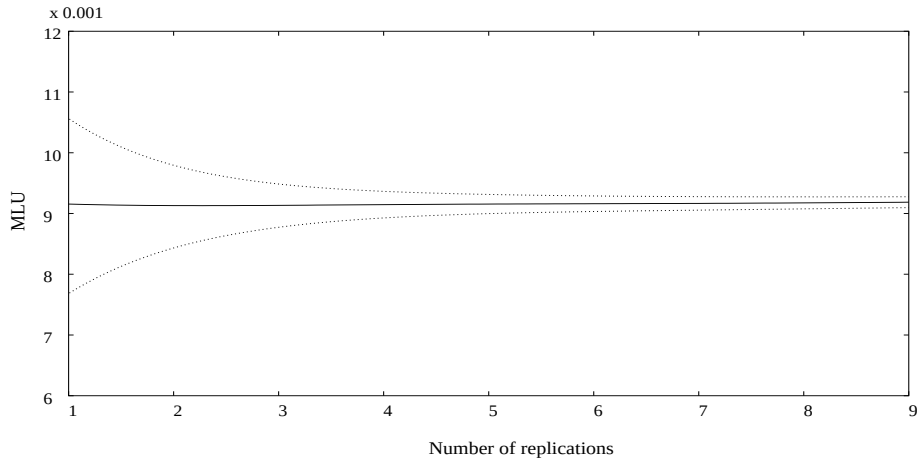


Figure 6.7: Confidence interval of HybridSA on Europe network

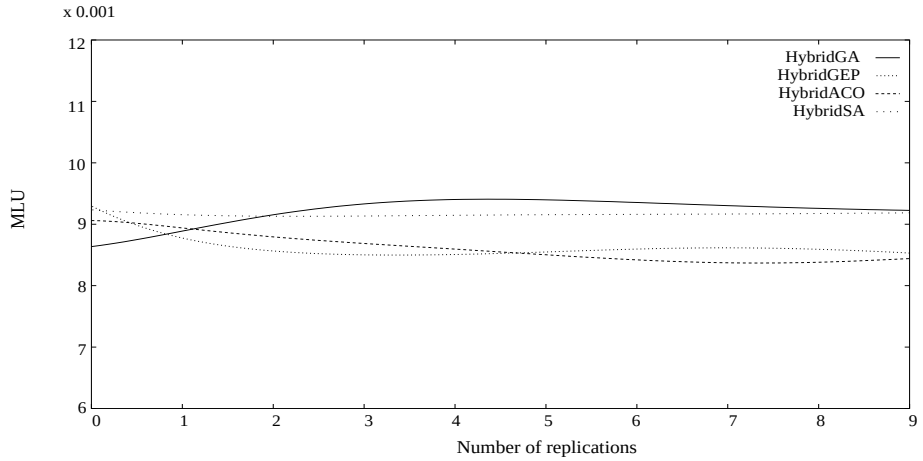


Figure 6.8: Comparison of cumulative means on Europe network

- HybridGA performs worse than HybridGEP.
- In most experiments, OSPF and HybridSA perform worse in terms of MLU while still achieving lower ALU.

Note however that the relative efficiency of OSPF and HybridSA in terms of ALU is usually balanced by lower throughput when deploying the link weights computed by these two algorithms to achieve IGP TE.

To assess the robustness of our results, we conducted similar experiments by scaling the demands $d_{i,e}$ of our initial traffic matrix $D = \{d_{i,e}\}$ by a parameter $k \geq 1$ to simulate

high bandwidth demanding applications using the traffic matrix $\tilde{D} = \{kd_{i,e}\}$. The results obtained are not reported in this thesis since they revealed similar performance patterns where HybridACO outperform the other algorithms and HybridSA and OSPF perform worse.

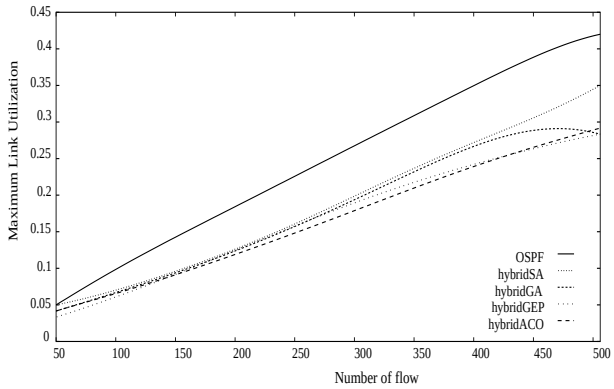


Figure 6.9: Comparison of Maximum Link Utilization

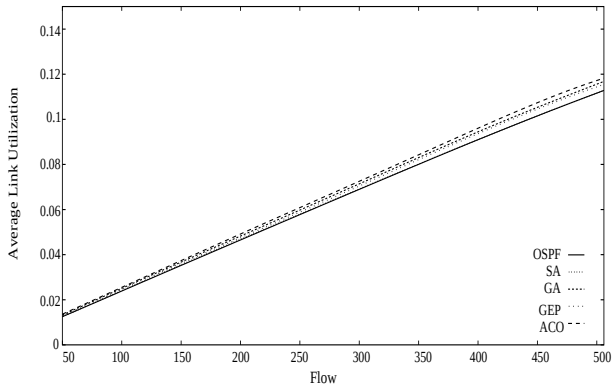


Figure 6.10: Comparison of Average Link Utilization

The results above are in agreement with those presented by Figure 6.9 and Figure 6.10 where the comparison of the MLU and ALU values computed by the different algorithms under different traffic profiles reveal that

- OSPF always achieves the highest MLU value than other algorithms but always the least ALU.
- HybridSA performs worse than the other three hybrid algorithms. Its smooth curve reveal that it reacts to the change of traffic demands without frequently adjusting its parameters.

- HybridACO mostly achieves lower MLU than the other algorithms. HybridACO reacts to the changes of traffic demands.
- HybridGA and HybridGEP are not as adaptive as HybridACO. There is room to improve these algorithms by changing their parameters through adaptive operations to adapt to the traffic demands.
- HybridGEP mostly achieves lower MLU than HybridGA.

6.4 The impact of parameter setting on performance

We conducted another set of experiments to compare the impact of parameter adjustment on the performance achieved by the different algorithms when finding a link weight assignment for the USA network under average traffic profile where traffic is offered to the network on 250 O-D pairs. The experimental results presented by Table 6.2 are averages computed over 20 replications to achieve 95% confidence interval.

HybridSA. We compared the results computed by $HybridSA$ using four different sets of parameters. Table 6.2 shows that neither increasing the number of loops from $loop = 50$ to $loop = 200$ or increasing the initial temperature from $T0 = 100$ to $T0 = 10000$ lead to better solution. Since the α value is only associated to the scale of the used bandwidth, it is not necessary to be adjusted. Because the result ($MLU = 0.15$) is not the global optimum, it could be improved by using appropriate parameter values. However, this was not tested in our experiments.

HybridGA. We compared the results computed by $HybridGA$ using four different sets of parameters where the number of generations is changed from $G = 10$ to $G = 60$. The results in Table 6.2 show that increasing the number of generations does not improve the MLU although the ALU are changed. Using a larger population size in combination with a higher number of generations could probably improve the MLU but result in higher processing overheads which might not be acceptable in practice for scalability reasons. Playing with different other parameters such as the probabilities associated with the genetic operations and using more complex genetic operations would also likely improve the MLU. However such improvements are beyond the scope of this thesis work.

HybridSA	**MLU**	**ALU**
T0=100,Loop=50, $\alpha = 100$	0.15	0.058
T0=100,Loop=100, $\alpha = 100$	0.15	0.058
T0=100,Loop=200, $\alpha = 100$	0.15	0.058
T0=10000,Loop=50, $\alpha = 100$	0.15	0.058
HybridGA	**MLU**	**ALU**
POPU=60,G=10	0.158	0.0607
POPU=60,G=20	0.158	0.0579
POPU=60,G=30	0.158	0.0607
POPU=60,G=60	0.158	0.0607
HybridGEP	**MLU**	**ALU**
POPU=60,G=10	0.175	0.0580
POPU=60,G=20	0.167	0.0580
POPU=60,G=30	0.142	0.0582
HybridACO	**MLU**	**ALU**
10 ants	0.158	0.0615
100 ants	0.142	0.0621
150 ants	0.142	0.0621
200 ants	0.142	0.0621

Table 6.2: Impact of parameter setting

HybridGEP. We compared the results computed by *HybridGEP* using three different sets of parameter values. The results in Table 6.2 show that the MLU is improved with the number of generations.
HybridACO. We compared the results computed from *HybridACO* using four different sets of parameter values where the number of ants are increased from 10 to 200. The results in Table 6.2 show that the MLU is improved when an adjustment is done from 10 to 100 ants and reveal that further adjustments are useless since they keep the MLU to the same value $MLU = 0.142$.

6.5 Conclusion

We compared in this chapter the different computational intelligence algorithms using different test networks under different traffic profiles. The experimental results presented above revealed that:

* All hybrid algorithms present a lower MLU but higher ALU compared to the OSPF algorithm.

* HybridSA performs worse than other hybrid algorithms since as shown earlier it finds difficult to jump out of a local optimum and it has hard to improve its MLU by adjusting its parameters.

* Among all the four hybrid algorithms, HybridSA is the simplest and the fastest algorithm. HybridACO takes the second position in terms of processing time while HybridGA and HybridGEP perform similarly because they come from the same family.

* HybridGEP is faster than HybridGA to converge to its optimal value. The reason is that HybridGEP has more complex genetic operations leading to a wider search space which increases its chance to get a better solution than HybridGA.

* On most of our test networks, HybridACO achieves the best result on several performance parameters. This is in agreement with previous results [26] which revealed

that it reacts better to the change of network topology and traffic demands. Some of the other three hybrid algorithms work well for the purpose of minimizing the MLU and could be improved by carefully adjusting their parameter values. However these adjustments have not been considered in this thesis work since they fall beyond the scope of the intended research purpose.

* HybridGA and the HybridGEP are more sensitive to the change of traffic demands than HybridACO and HybridSA.

* Considering its relative simplicity and good results, HybridACO performs the best over all the hybrid algorithms.

Chapter 7

Conclusion

7.1 Summary

This thesis work has described and applied four computational intelligence algorithms on a set of test networks under different traffic profiles to achieve IGP traffic engineering. These include (1) Genetic Algorithm (2) Gene Expression Programming (3) Ant Colony optimization and (4) Simulated Annealing. We have used a combination of global and local search to find a set of link weights that minimize the Maximum Link Utilization in IGP routing. We have tested our algorithmic solutions using a two-layer routing process where link weights are calculated using an offline model while the NS simulator is used in a second step to route the traffic offered to a network using the link weights calculated by the offline process. Using four test networks, we compared the results obtained from our algorithms in terms of Maximum Link Utilization and the Average Link Utilization and analyzed the impact of different design parameters and operators on the scalability and performance of the IGP TE procedure. The results revealed that while our four computational algorithms can be used to improve IGP routing and outperform classic OSPF routing, the HybridACO algorithm performs better than the others.

7.2 Future work

The algorithms presented in our thesis could be extended in different ways:

* More work need to be done using different topologies and traffic profiles to assess the routing scalability in terms of routing simplicity and fast convergence to the global optimum.

* The relationship between the algorithm's parameters and the network topology is very important for practical use. It should be analyzed in details to improve the performance achieved by the IGP TE process.

* More routing scenarios including various network topologies, flow demands, and scheduling models should be considered in evaluating the algorithm's performance. More work need to be done to evaluate the impact of the change of network topology and flow demands on the algorithms.

* Extensions to concurrent computation is an approach to speed the convergence to the global optimum. These extensions can be studied in future work.

* Extensions to the case of multi-constraint, multi-objective, and equal cost multi-path routing are needed to be considered. They can also be studied in future work.

Bibliography

[1] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4). Technical RFC 1771, IETF Network Working Group*, 1995.

[2] J.T. Moy. *OSPF protocol analysis. Technical RFC 1245, IETF Network Working Group*, 1991.

[3] D. Oran. *OSI IS-IS Intra-domain Routing Protocol. Technical RFC 1142, IETF Network Working Group*, 1990.

[4] A. Riedl. *Optimized routing adaptation in IP networks utilizing OSPF and MPLS. Proceedings of IEEE ICC2003*, May 2003.

[5] A.B. Bagula. *Hybrid Routing in Next Generation IP Networks. Elsevier Computer Communications,* Volume 29, Number 7, Pages 879-892, April 2006.

[6] A. Feldmann et al. *Netscope: Traffic Engineering for IP networks. IEEE Network Magazine*, Volume 14, Pages 11-19, 2000.

[7] A. Feldmann et al. *Deriving Traffic demands for operational IP Networks: Methodology and experience. IEEE/ACM Transactions on Networking*, Volume 9, Pages 265-279, 2001.

[8] M. Ericsson, Mauricio G. C. Resende, and Panos M. Pardalos. *A Genetic Algorithm for the Weight Setting Problem in OSPF Routing.* J. Comb. Optim. 6(3): 299-333, 2002.

[9] T.M. Thomas II. *OSPF Network Design Solutions. Cisco Press*, 1998.

[10] Bernard Fortz and Mikkel Thorup. *Internet Traffic Engineering by Optimizing OSPF Weights. Proceedings of IEEE INFOCOM*, March 2000.

[11] M. Rodrigues and K.G. Ramakrishnan. *Optimal routing in data networks. International Telecommunication Synposium (ITS)*, 1994.

[12] A. Bley et al. *Design of baroadband virtual private networks: Model and heuristics for the B-WiN. Proc. DIMACS Workshop on Robust Communication Network and Survivability, AMS-DIMACS series*, 1998.

[13] F. Lin and J. Wang. *Minimax open shortest path first routing algorithms in network supporting the smds services. Proceedings of the 2006 IEEE International Conference on Communications (ICC), Volume 2*, Pages 666-670, 1993.

[14] Mattias Söderqvist. *Search Heuristics for Load Balancing in IP-networks. Swedish Institute of Computer Science, SICS Technical Report T2005:04 ISSN 1100-3154*, ISRN:SICS-T–2005/04-SE.

[15] M. Piro et al. *On open shortest path first related network optimisation problems. Performance Evaluation archive Volume 48, Issue 1-4*, Pages: 201 - 223, 2002.

[16] A. Riedl. *A Hybrid Genetic Algorithm for Routing Optimization in IP Networks Utilizing Bandwidth and Delay Metrics.* October 2002.

[17] U. Killat E. Mulyana. *A Hybrid Genetic Algorithm Approach For OSPF Weight Setting Problem. Proceedings of the 9th Polish Teletraffic Symposium*, 2002.

[18] L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. *A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks, Volume 46, Issue 1* Pages 36 - 56, June 2005.

[19] L.S. Buriol, C.C. Ribeiro G.C Resende, and M. Thorup. *A memetic algorithms for OSPF routing. Proceedings of the 6th INFORMS Telecom*, Pages 187-188, 2002.

[20] C. Ferreira. *Gene Expression Programming: A New Adaptive Algorithm for solving problems. Complex Systems, Volume 13, Number 2, Pages 87-129, 2001.*

[21] A.B. Bagula and H. Wang. On the Relevance of Using Gene Expression Programming in Destination-based Traffic Engineering. *Lecture Notes in Computer Sciences, Volume 3801, Pages 224-229, December 2005.*

[22] A.B. Bagula. Traffic Engineering Next Generation IP Networks Using Gene Expression Programming. *Proceedings of the 2006 IEEE/IFIP Network Operations & Management Symposium, Pages 230-239, April 2006.*

[23] R. Schoonderwoerd et al. Ant-based load balancing in telecommunication networks. *Adaptive Behaviour, 5(2), Pages 169-207, 1997.*

[24] G. Di Caro and M. Dorigo. Mobile agents for adaptive routing. *Proc. 31st Hawaii Intl. Conf. systems Scienecs (HICSS-31), Kohala coast, Hawaii, Pages 74-83, Jan 1998.*

[25] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research, JAIR,9, Pages 317-365, 1998.*

[26] S.S. Dhillon and P. Van Mieghen. Performance analysis of the AntNet algorithm. *Computer Networks, Vol 51, Pages 2104-2125, 2007.*

[27] Cisco. Configuring OSPF. *Cisco Press, 1997.*

[28] Andries P. Engelbrecht. Computational Intelligence: An Introduction. *John Wiley & Sons, Ltd, ISBN 0-470-84870-7, 2005.*

[29] Network Simulator. http://www.isi.edu/nsnam/ns/.

[30] A. Colorni et al. Distributed Optimization by Ant Colonies. *Proc. Fisrt European Conf. on Artificial Life (ECAL91), Paris, France, Pages 134-142, 1991.*

[31] M. Dorigo et al. Ant Algorithms for Discrete Optimization. *Artificial Life, 5,2, Pages 137-171, 1999.*

[32] E. Bonabeau and E. Theraulaz. Swarm Smarts. *Scientific American, Pages 55-61, March 2000.*

[33] James Kennedy and Russell C. Eberhart. Swarm Intelligence. *Morgan Kaufmann Publishers, ISBN 1-55860-595-9, 2001.*

[34] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science 220, Pages 671-680, 1983.*

[35] Stewart Robinson. Simulation: The Practice of Model Development and Use. *John Wiley & Sons, Ltd, ISBN 0-470-84772-7.*