UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA,
INFORMÁTICA INDUSTRIAL E TELEMÁTICA

JOSÉ RICARDO HOFFMANN

GENIUS - UM ESCALONAMENTO BASEADO EM ALGORITMOS GENÉTICOS PARA COMUTADORES DE ALTO DESEMPENHO

DISSERTAÇÃO

CURITIBA

2013

JOSÉ RICARDO HOFFMANN

GENIUS - UM ESCALONAMENTO BASEADO EM ALGORITMOS GENÉTICOS PARA COMUTADORES DE ALTO DESEMPENHO

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica, Informática Industrial e Telemática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Ciências" – Área de Concentração: Telemática.

Orientador: Emilio Carlos Gomes Wille

CURITIBA

2013

Dados Internacionais de Catalogação na Publicação

H711 Hoffmann, José Ricardo

Genius : um escalonamento baseado em algoritmos genéticos para comutadores de alto desempenho / José Ricardo Hoffmann. — 2013.

55 f. : il. ; 30 cm

Orientador : Emilio Carlos Gomes Wille.

Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica, Informática Industrial e Telemática. Área de concentração: Telemática. Curitiba, 2013.

Bibliografia: p. 52-53.

1. Roteadores (Redes de computação). 2. Mecanismo de distribuição elétrica. 3. Agenda de execução (Administração). 4. Algoritmos genéticos. 5. Engenharia elétrica – Dissertações. I. Wille, Emilio Carlos Gomes, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

Biblioteca Central da UTFPR, Campus Curitiba

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº. 645

"Genius – Um escalonamento baseado em algoritmos genéticos para comutadores de alto desempenho."

por

José Ricardo Hoffmann

Orientador: Prof. Dr. Emilio Carlos Gomes Wille

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Telecomunicações e Redes do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR, às 9h30 do dia 18 de outubro de 2013. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores doutores:

Prof. Emilio Carlos Gomes Wille, Dr.
(Presidente – UTFPR)

Prof. Eduardo Parente Ribeiro , Dr.
(UFPR)

Profª. Myriam Regattieri de Biase da Silva Delgado, Dr.
(UTFPR)

Visto da coordenação:

Prof. Ricardo Lüders, Dr.
(Coordenador do CPGEI)

AGRADECIMENTOS

Agradeço primeiramente a Deus pelas oportunidades que me são concedidas e em momentos difíceis por proporcionar amparo.

Agradeço a minha mãe pelo ensino, apoio durante toda a minha caminhada.

Principalmente ao meu professor orientador, que com muito respeito, paciência, compreensão, profissionalismo e conhecimento proporcionou a possibilidade de conclusão desse trabalho.

E a todas as pessoas que participaram de alguma forma dessa minha caminhada.

RESUMO

HOFFMANN, José Ricardo. GENIUS - UM ESCALONAMENTO BASEADO EM ALGORITMOS GENÉTICOS PARA COMUTADORES DE ALTO DESEMPENHO. 55 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica, Informática Industrial e Telemática, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Um dos mais importantes elementos que compõem uma rede de telecomunicações é o roteador. Os roteadores modernos empregam sofisticados comutadores para a transmissão de pacotes. A arquitetura de comutadores com filas de entrada exige um processo de escalonamento que estabelece a transferência de pacotes das portas de entrada às portas de saída. O desempenho do sistema depende diretamente do algoritmo de escalonamento, considerando sua vazão e complexidade. Esta dissertação realiza o levantamento teórico dos algoritmos de escalonamento mais relevantes e propõe uma abordagem de escalonamento usando algoritmos genéticos. Um simulador baseado em eventos discretos foi desenvolvido para a realização de testes de desempenho dos escalonadores estudados. O algoritmo proposto, denominado GENIUS, apresentou desempenho relevante e baixa complexidade.

Palavras-chave: Roteadores, Comutador com filas de entrada, Algoritmo de escalonamento, Algoritmos Genéticos.

ABSTRACT

HOFFMANN, José Ricardo. GENIUS - A SCHEDULING BASED ON GENETIC ALGORITHMS FOR HIGH PERFORMANCE SWITCHES. 55 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica, Informática Industrial e Telemática, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

One of the most important components of network telecommunications is the router. Modern routers employ input-queued crossbar switches that require sophisticated scheduling techniques for packet transmission. The architecture of switches with input queues requires an scheduling process that establishes the transfer of packets from input to output ports. The performance of router depends of the scheduling algorithm, considering its throughput and complexity. In this work we survey the most relevant theoretical scheduling algorithms and propose a scheduling approach using genetic algorithms. We developed a simulator of discrete events for testing of schedulers performance. The proposed algorithm, called GENIUS, presents relevant performance and low complexity.

Keywords: Router, Input-Queued switches, Scheduling algorithms, Genetic Algorithms.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CPU	Central Processing Unit
WFQ	Weighted Fair Queuing
QoS	Quality of Service
ATM	Asynchronous Transfer Mode
VOQ	Virtual Output Queuing
HoL	Head of the Line
VOQ	Virtual Output Queuing
MWM	Maximum Weighted Matching
APSARA	A Parallel Scheduling Algorithm and its Random Aproximation
LQF	Longest Queue First
RPA	Reservation with Preemption and Acknowledgment
RRM	Round Robin Matching
AGs	Algoritmos Genéticos
AG	Algoritmo Genético
GENIUS	Genetic Scheduling Algorithm for High-Performance Switches
OX	Order Crossover
SIM	Simple Inversion Mutation

LISTA DE SÍMBOLOS

N	número de portas
M	célula / pacote
i	porta de entrada
j	porta de saída
VOQ	representação de fila
t	instante de tempo
q	dimensão de fila
Q	matriz de pesos
λ	taxa média de chegada
x	conexão entre portas
S	conjunto de matchings
X	matching
W	peso do matching
K	tamanho da população
O	complexidade
N_I	número de indivíduos
p_c	probabilidade de crossover
p_m	probabilidade de mutação
ρ	carga normalizada
$E[L]$	tamanho médio de fila
$E[T]$	atraso médio de atendimento
k	tamanho do torneio

SUMÁRIO

1 INTRODUÇÃO

A Internet é uma rede de comutação de pacotes, construída em torno de uma grande variedade de sistemas de transmissão e comutação. Os sistemas de comutação mais importantes da Internet são os roteadores. As principais funções de um roteador são receber os pacotes das portas de entrada, encontrar o destino de porta de saída adequado com base na tabela de encaminhamento e realizar a transfência dos pacotes. Estas duas funções básicas, roteamento e comutação, são difíceis de serem implementadas quando a taxa de transferência exigida é muito alta, uma vez que algoritmos complexos devem ser executados em um espaço de tempo muito curto (KUROSE, 2006).

O desenvolvimento de roteadores de alto desempenho é particularmente importante, dado que a Internet hoje é composta por um número relativamente pequeno de redes dorsais muito rápidas, que interconectam um número muito grande de redes menores.

Os comutadores (*switches*) possuem uma arquitetura composta por *buffers* e pelo elemento de comutação (*switch fabric*). Para arquiteturas de comutadores de alto desempenho com *buffers* nas portas de entrada, o principal gargalo de desempenho é dado pelo algoritmo de escalonamento, que seleciona pacotes a serem transferidos pelo elemento de comutação.

1.1 MOTIVAÇÃO

Os algoritmos de escalonamento, em roteadores, têm o papel de transferir pacotes das portas de entrada para portas de saída de acordo com a tabela de encaminhamento. Em algumas arquiteturas de roteadores (especialmente em comutadores com *buffers* nas portas de entrada) esse algoritmo pode garantir uma transferência de 100% sem perda de pacotes. Essa transferência garante um maior desempenho das redes de comunicações visto que não haverá perdas durante as transmissões (GIACCONE, 2002).

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

Os algoritmos genéticos (AGs) são exemplos de métodos de computação evolucionária usados para busca, otimização e aprendizado de máquina. Nesta dissertação, suas características naturais de busca em ambientes dinâmicos (*changing environments*)(GREFENSTETTE, 1992) são exploradas na construção de um algoritmo de escalonamento denominado GENIUS (*Genetic Scheduling Algorithm for High-Performance Switches*). Duas versões são propostas e analizadas: a versão elitista GENIUS-E, visando alcançar um desempenho próximo ao ótimo, e a versão simplificada GENIUS-S de complexidade reduzida.

1.2.2 OBJETIVOS ESPECÍFICOS

- Levantamento teórico de todas as bases envolvidas;
- Desenvolvimento de um novo escalonador baseado em algoritmos genéticos;
- Desenvolvimento de um simulador controlado a eventos discretos específico para análise de desempenho do sistema;
- Fornecer resultados de desempenho utilizando diferentes padrões de tráfego;
- Comparar o desempenho do escalonador desenvolvido com escalonadores relevantes.

1.3 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada da seguinte forma. O Capítulo 2 apresenta a arquitetura e o funcionamento do comutador bem como a notação matemática adotada. O Capítulo 3 define os algoritmos de escalonamento relevantes e relacionados ao proposto, e seus respectivos funcionamentos. A revisão de algoritmos genéticos é descrita pelo Capítulo 4. O Capítulo 5 descreve detalhadamente o escalonador proposto e suas versões. No Capítulo 6, resultados numéricos são apresentados e discutidos. As conclusões são apresentadas no Capítulo 7.

2 ARQUITETURA DE ROTEADORES

Uma estratégia para tratar o crescimento exponencial do tráfego na Internet é o desenvolvimento de um roteador com maior capacidade de comutação. Este capítulo descreve as principais características e arquiteturas envolvidas no desenvolvimento de comutadores de alto desempenho, levando em conta sua disposição dentro da rede.

Um roteador é composto basicamente por portas de entrada, elemento de comutação e portas de saída. As portas de entrada realizam funções de camada física e enlace, o elemento de comutação conecta as portas de entrada do roteador às portas de saída. As portas de saída recebem os pacotes que foram repassados através do elemento de comutação e então, os transmite até o enlace de saída(KUROSE, 2006).

Embora existam abordagens de diferentes projetos de roteadores envolvendo várias categorias, os principais elementos de um roteador genérico são(GIACCONE, 2002):

- Processador de roteamento: gerencia todo o processo de roteamento, computando a tabela de roteamento repassada e determinada pelo protocolo de roteamento;
- Portas de entrada e saída: recebem e transmitem pacotes pelos canais de comunicação, providenciam o encapsulamento e desencapsulamento de dados, ainda podem classificar pacotes seguindo regras especificadas pelo processador de roteamento;
- Mecanismo de transmissão: Este módulo é opcional, destinado a encontrar rapidamente a porta de destino de um pacote, com base no endereço de destino e de alguma classificação;
- Elemento de comutação: transfere pacotes de uma porta a outra, conforme informações derivadas do mecanismo de escalonamento.

2.1 TÉCNICAS DE COMUTAÇÃO

O elemento de comutação é o principal elemento de um roteador, é por meio dele que os pacotes são repassados de uma porta à outra. A comutação pode ser realizada das seguintes

maneiras (KUROSE, 2006)(GIACCONE, 2002):

- Comutação por memória: os primeiros rotaedores eram computadores tradicionais onde a comutação era realizada diretamente sob o controle da CPU (processador de roteamento), as portas funcionavam como dispositivos tradicionais de entrada/saída. O funcionamento ocorria através da sinalização da porta de entrada, na qual a chegada de um pacote, sinalizava ao processador de roteamento por meio de uma interrupção. O pacote então era copiado para a memória do processador, então o processador extraía o endereço de destino do cabeçalho, consultava a porta de saída definida pela tabela de repasse e copiava o pacote para o *buffer* da mesma. Muitos roteadores modernos ainda comutam por memória, contudo, uma importante diferença entre esses roteadores e os antigos é que a consulta do endereço de destino e o armazenamento do pacote na localização adequada da memória são realizados pelo próprio comutador. Em certos aspectos, roteadores que comutam por memória se parecem muito com multiprocessadores de memória compartilhada, nos quais processadores de uma linha comutam pacotes para o *buffer* da porta de saída adequada. A Figura 1 ilustra essa técnica.

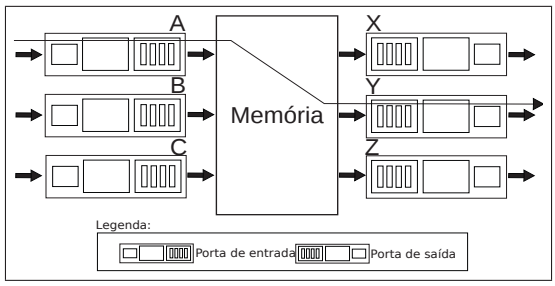

Figura 1: *Técnica de comutação por memória.*
Fonte: (KUROSE, 2006)

- Comutação por barramento: nessa arquitetura, as portas de entrada transferem um pacote diretamente para a porta de saída através de um barramento compartilhado sem a intervenção do processador de roteamento. Embora o processador de roteamento não esteja envolvido na transferência por barramento, somente um pacote por vez pode ser transferido, devido a esse compartilhamento do barramento, como mostra a Figura 2. Então como um único pacote deve atravessar o barramento por vez, a velocidade de comutação do roteador fica limitada à velocidade do barramento.

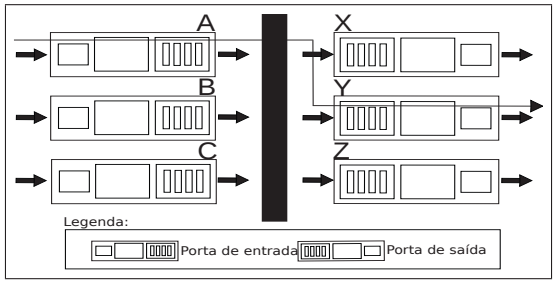

Figura 2: *Técnica de comutação por barramento.*
Fonte: (KUROSE, 2006)

- Comutação por uma rede de interconexão: o modo mais simples do *crossbar* é uma correspondência de N entradas (e, então, o número de saídas também seria N), o número de junções ou pontos de cruzamento é N^2. Como esses pontos de cruzamento podem estar ligados ou desligados, o dispositivo necessita de um algoritmo de escalonamento que indique quais entradas, em um instante específico, estão conectadas às saídas. Uma maneira de driblar a limitação de velocidade de um único barramento compartilhado é usar uma rede de interconexão mais sofisticada, tal como as que eram utilizadas no passado para interconectar processadores em uma arquitetura de computadores multiprocessados. Portanto um comutador do tipo *crossbar* é uma rede de interconexão que consiste em $2N$ barramentos que conectam N portas de entrada com N portas de saída, como mostra a Figura 3. Um pacote que chega a uma porta de entrada percorre o barramento horizontal ligado à porta de entrada até interceptar o barramento vertical que leva à porta de saída desejada. Se o barramento vertical que leva à porta de saída correspondente estiver livre, o pacote será transferido, do contrário o pacote ficará bloqueado e deverá entrar na fila da porta de entrada, dependendo posteriormente do algoritmo de escalonamento.

Em alguns roteadores, é na porta de entrada que o roteador determina para qual porta de saída o pacote será repassado através do elemento de comutação. A escolha da porta de saída é realizada utilizando a informação contida na tabela de encaminhamento. Com cópias locais da tabela de encaminhamento, as decisões podem ser tomadas sem a necessidade de consultar o processador de roteamento centralizado.

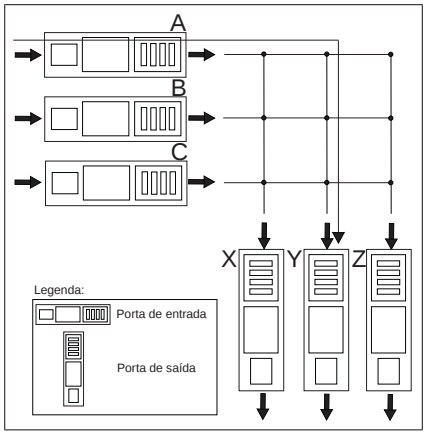

Figura 3: *Técnica de comutação por crossbar.*
Fonte: (KUROSE, 2006)

Dada a existência de uma tabela de encaminhamento, o repasse de pacotes consiste em procurar o registro de entrada compatível com o endereço de destino. Porém, na prática as coisas não são tão simples. Talvez o principal fator de complicação seja o fato de os roteadores de *backbone* operarem em altas velocidades, executando milhões de exames por segundo. Devido a necessidade de operar em altas velocidades por conta de enlaces atuais, é impossível uma busca linear em uma tabela de encaminhamento consideravelmente grande.

Os roteadores convencionais com filas nas portas de saída possuem um escalonador que deve escolher a transmissão para pacotes que estão nas portas de entrada, essa seleção pode ser feita com base em uma regra simples baseada na própria ordem da fila ou por uma regra de escalonamento mais sofisticada, tal como a fila ponderada justa WFQ (*Weighted Fair Queuing*), que compartilha o enlace de saída com otimização entre as diferentes conexões fim-a-fim. O escalonamento de pacotes desempenha um papel crucial no fornecimento de garantia de qualidade de serviço (*QoS*) (KUROSE, 2006).

Projetos de roteadores com *buffers* de saída são populares pois fornecem um desempenho ideal. No entanto, esse projeto pode exigir um aumento de velocidade de comutação, ou seja, a taxa de transferência interna de dados deve ser superior a velocidade de um enlace externo. As taxas de velocidades de dados tendem a crescer rapidamente, através de enlaces de fibra principalmente, proporcionando dentro dos roteadores uma aceleração de comutação necessária para o *buffer* de saída, e isso torna-se cada vez mais difícil. Por essa razão, projetos de roteadores com filas de entrada têm recebido muita atenção recentemente.

2.2 ROTEADORES COM FILAS DE ENTRADA

A Figura 4 mostra a estrutura lógica de um comutador com filas de entrada. O comutador opera em unidades de dados de tamanhos fixos, que podem ser células *ATM* (*Asynchronous Transfer Mode*), ou ter qualquer outro formato conveniente. Utilizando o termo *ATM*, as células identificam as unidades de dados de tamanho fixo. Na verdade, os resultados não dependem de que as células utilizadas possuam tamanho fixo, mas referem-se à tomada de decisão de comutação em instantes igualmente espaçados. A distância entre duas decisões de comutação é denominada *time slot* que representa a granularidade na alocação de recursos do roteador.

Considera-se um comutador com N entradas e N saídas. Assume-se que todas as linhas de entrada e saída atuam com a mesma velocidade. Os pacotes são armazenados nas interfaces de entrada. Cada entrada possui e gerencia uma fila para cada saída, portanto, um total de $N \times N$ filas são necessárias. Cada fila pode armazenar até M células e pacotes em excesso são descartados. Esta separação de filas, chamada *VOQ (Virtual Output Queuing)*, permite evitar degradações de desempenho devido ao bloqueio *HoL (Head of the Line)* (KAROL et al., 1987).

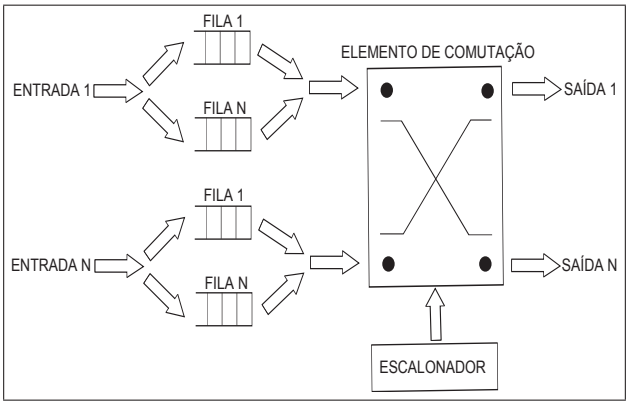

Figura 4: *Estrutura lógica de um comutador com filas de entrada.*

O elemento de comutação é sem bloqueio, sem memória, e sem atraso: no máximo, uma célula pode ser removida de cada entrada e, no máximo, uma célula pode ser transferida para cada uma das saídas em cada *time slot*. O algoritmo de escalonamento decide quais pacotes serão transferidos, a partir das entradas, para as saídas do comutador em cada *time slot*.

2.3 DEFINIÇÃO DO PROBLEMA DE ESCALONAMENTO

Os algoritmos de escalonamento em questão evitam o bloqueio e resolvem a contenção dentro da estrutura de comutação. Assume-se um esquema de *VOQ - Virtual Output Queuing*. O enfileiramento do pacote da entrada i para a saída j é chamado de VOQ_{ij} e esta ocupação é representada por X_{ij} (GIACCONE, 2002).

Esta dissertação considera um comutador com $N \times N$ filas de entrada. O *buffer* na entrada i é particionado em N "filas virtuais de saída" ($VOQs$), onde VOQ_{ij} armazena pacotes na entrada i para a saída j. Denota-se a dimensão de VOQ_{ij} no instante t por $q_{ij}(t)$. Sendo $Q(t) = [q_{ij}(t)]$ uma matriz $N \times N$ contendo a dimensão de todas as filas $VOQs$ no instante t.

Seja λ_{ij} a taxa média na qual os pacotes chegam na entrada i para a saída j, e $\Lambda = [\lambda_{ij}]$ a matriz de taxa de chegada, também chamada matriz de carga. Exige-se uma carga admissível, ou seja, $\sum_j \lambda_{ij} \leq 1$ para todo i, e $\sum_i \lambda_{ij} \leq 1$ para todo j. Em outras palavras, esta situação assegura que nenhuma entrada ou saída esteja sobrecarregada. A carga máxima de entrada é dada por $\rho = max_i(\sum_j \lambda_{ij})$.

Utilizam-se variáveis binárias $x_{ij}(t), i, j = 1, \ldots, N$, para representar conexões. A entrada i está conectada com a saída j no momento t, se e somente se, $x_{ij}(t) = 1$. Sem perda de generalidade, consideram-se apenas ligações completas, isto é, permite-se uma conexão entre a entrada i e saída j, mesmo se $q_{ij}(t) = 0$.

Uma configuração de conexão admissível pode ser vista como um casamento (*matching*) em um grafo bipartido representado pela Figura 5. Entradas e saídas correspondem aos nós do grafo, e uma aresta entre entrada i e saída j indica uma conexão. Seja $X(t) = [x_{ij}(t)]$ a matriz de matching no instante t.

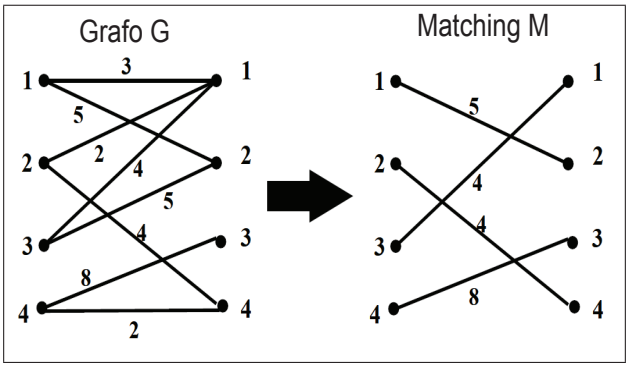

Figura 5: *Problema de escalonamento descrição do grafo bipartido.*

Fonte: (GIACCONE, 2002)

Define-se o peso do matching $X(t) = [x_{ij}(t)]$ como $W(t) = \sum_{ij} q_{ij}(t)x_{ij}(t)$, onde o peso da aresta entre a entrada *i* e a saída *j* é igual ao tamanho da fila $q_{ij}(t)$. Para um comutador $N \times N$, o conjunto de todos os possíveis matchings, indicados por S, tem cardinalidade $N!$.

O algoritmo de escalonamento seleciona um *matching* X, um conjunto de pares de entrada e saída sem conflito, de tal modo que cada entrada é ligada no máximo com uma saída, e tal saída é ligada no máximo com uma entrada. Em cada *time slot*, se a entrada *i* é conectada com a saída *j*, uma célula é removida da fila VOQ_{ij}, e tranferida para a saída *j*, configurando uma a estrutura de comutação sem bloqueio.

3 ALGORITMOS DE ESCALONAMENTO

A grande concorrência dentro das redes de comunicação origina situações de competição pela utilização de recursos. Em redes que suportam integração de serviços, torna-se necessário providenciar QoS diferenciado para categorias de serviço (ou classes de tráfego), oferecendo garantias de desempenho a aplicações críticas e ao mesmo tempo permitindo um compartilhamento de recursos de acordo com critérios estabelecidos(KUROSE, 2006).

Os algoritmos de escalonamento são componentes essenciais para garantir o desempenho sob a concorrência, uma vez que determinam a ordem dos fluxos que passarão pela rede. Algoritmos de escalonamento realizam duas funções principais: decidem a ordem de serviço de fluxos em competição e gerenciam as filas de solicitação de serviço.

Esses algoritmos são utilizados em qualquer sistema ou camada de protocolo em que haja contenção por recursos, mas em especial em elementos de rede (roteadores, comutadores, etc). Desempenham um papel importante na provisão de diferentes níveis de QoS e em diferentes aplicações, permitindo o controle diferenciado de atraso, largura de banda e taxa de perda. Aplicações com requisitos de tempo real necessitam de garantias absolutas, já em aplicações que não necessitam quaisquer garantias, é desejável que o compartilhamento de recursos seja realizado de forma justa (KUROSE, 2006).

Tais algoritmos são referenciados como estáveis quando atingem 100% de transferência do tráfego ofertado, ou seja, quando não ocorrem perdas de pacotes durante o processo de transferência (GIACCONE et al., 2001).

É função do algoritmo de escalonamento determinar, em cada momento t, o *matching* particular a ser utilizado, com base na seleção das maiores filas, garantindo que a contenção não ultrapasse a capacidade de armazenamento do sistema.

3.1 MWM

Um algoritmo de escalonamento de interesse particular é o MWM (*Maximum Weighted Matching*). O escalonador MWM escolhe, em cada instante t, o matching de maior peso. Mais precisamente, se $X^w(t)$ representa o matching determinado pelo MWM no instante t, então $X^w(t)$ é dado por:

$$X^w(t) = arg\ max_{X \in S_n} \{\sum_{i,j} x_{ij} q_{ij}(t)\} \quad (1)$$

O algoritmo de escalonamento MWM garante 100% de transferência (*throughput*) para todos os padrões de tráfego (Bernoulli) admissíveis de entrada (TASSIULAS; EPHREMIDES, 1992), (MCKEOWN et al., 1996). A literatura mostra que o escalonador MWM apresenta valores baixos de atraso médio (por manter tamanhos de fila pequenos), entretanto, por ser o problema resolvido mediante aplicação do algoritmo Húngaro (KUHN, 2005), apresenta complexidade temporal $O(N^3)$.

No entanto esse algoritmo retorna um resultado ótimo quando aplicado, o que faz com que todos os novos resultados obtidos com outros algoritmos de escalonamento sejam comparados com ele, almejando alcançar seus resultados.

3.2 APSARA

O algoritmo de escalonamento APSARA (*A Parallel Scheduling Algorithm and its Random Aproximation*) foi proposto em (GIACCONE et al., 2001), e por ser baseado em uma população de soluções é tomado como base de comparação neste trabalho.

Seja $X(t)$ o matching determinado pelo algoritmo APSARA no instante t, $\mathcal{N}[X(t)]$ o conjunto de vizinhos do matching $X(t)$ (cada vizinho difere do matching $X(t)$ em exatamente duas arestas) e seja $Q(t+1)$ o tamanho de fila no início do instante $t+1$. No instante $t+1$, o APSARA procede da seguinte forma:

- Determina os vizinhos, $\mathcal{N}[X(t)]$, de $X(t)$ e o matching $Z(t+1)$ que corresponde a um caminho hamiltoniano[1] no instante $t+1$.
- Seja $S(t+1) = \mathcal{N}[X(t)] \cup Z(t+1) \cup X(t)$. Calcula o peso de cada matching $Y \in S(t+1)$ como $W(Y) = \sum_{ij} y_{ij} q_{ij}(t+1)$.

[1] Caminho que permite passar por todos os nós de um grafo exatamente uma única vez(LUGER, 2004).

- Determina o matching no instante $t+1$ por $X^w(t+1) = arg\ max_{U \in S_{(t+1)}}\{W(U)\}$.

O algoritmo básico APSARA-B, requer o cálculo do peso para os matchings vizinhos. Esse cálculo é simples, porque um vizinho Y difere do matching $X(t)$ em exatamente duas arestas. No entanto, o cálculo dos pesos de todos os $\binom{N}{2}$ vizinhos, se for feito em paralelo como mostrado na Figura 6, necessitará de grande quantidade de espaço em hardware para grandes valores de N.

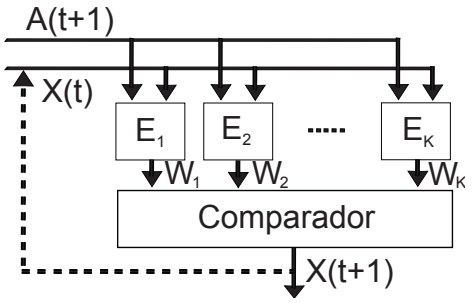

Figura 6: Esquema para implementação do APSARA.
Fonte: (GIACCONE et al., 2001)

Nota-se que, para a construção de comutadores com grande número de portas, o número de módulos em paralelo faria o sistema proibitivamente caro. Uma solução consiste em manter um máximo de $K\ (<< N^2)$ vizinhos escolhidos de forma aleatória a partir do conjunto $\mathcal{N}[X(t)]$. Essa consideração dá origem ao algoritmo randomizado APSARA-R(SHAH et al., 2002)(GIACCONE et al., 2003).

O grande ganho, neste caso, é o tempo (APSARA requer apenas uma iteração), e para elementos de comutação ligados a linhas de alta velocidade, o tempo disponível para o escalonamento é muito pequeno. Assim, o APSARA ajuda neste caso, trocando o espaço pelo tempo.

Giaccone et al (2003) levantam a seguinte questão, é possível para um algoritmo competir com o desempenho do MWM e ainda ser simples de implementar? Se sim, quais características do problema continuam a ser exploradas? A resposta está em reconhecer duas características do problema de escalonamento em comutadores de alta velocidade:

- Pacotes chegam (ou partem) com uma taxa de no máximo um por porta de entrada (ou saída) em um determinado *time slot*. As ocupações médias das filas mudam muito pouco durante uma sequência de *time slots*. Isto sugere que um matching bastante pesado

(matching que seleciona filas com muitas células a serem transferidas) continuará assim por mais alguns *time slots*.

- Dois matchings escolhidos aleatoriamente que diferem por pouco (por exemplo, em apenas duas arestas) muito provavelmente serão igualmente pesados. Isto é, dado um matching pesado X, existirá provavelmente um matching X' vizinho de X, que também é pesado. Isso fornece uma base bastante eficiente para a pesquisa do conjunto de matchings, sobre *time slots* sucessivos, pela observação os vizinhos do matching atual.

3.2.1 MAX-APSARA

O APSARA gera todos os matchings no conjunto de vizinhança independentemente do tamanho das filas. O tamanho das filas é usado somente para selecionar o *matching* mais pesado da vizinhança. Por este motivo, é possível que o *matching* determinado pelo APSARA seja o melhor dentre os vizinhos, mas não o *matching* ótimo. Isto é, existe uma entrada i, que possui dados para a saída j, mas o *matching* $X(t)$ conecta a entrada i para alguma outra saída j' e conecta a saída j para alguma outra entrada i', e ambos $q_{ij'}(t)$ e $q_{i'j}(t)$ são nulos. Assim, a entrada i e a saída j não estão sendo corretamente conectadas. Deve-se, então, completar o *matching* determinado pelo APSARA, transformando-o em um matching de peso máximo. Essa opção é chamada de Max-APSARA(GIACCONE, 2002).

A versão Max-APSARA utiliza como base o algoritmo APSARA a fim de criar seu *matching* inicial, desse matching, selecionam todas as conexões com pesos nulos criando um *submatching*. Para esse *submatching*, cria-se uma nova tabela contendo os tamanhos das filas e aplica um algoritmo de aproximação simples de correspondência ponderada (PREIS, 1999) que garante até 50% do desempenho ótimo com uma complexidade $O(N)$. Esse algoritmo tem o intuito de utilizar pesos não nulos, visando criar o *matching* utilizando os maiores valores da matriz de pesos, assim maximizando o peso total do *submatching* que antes era nulo. Ao final de todo o processo o escalonador encontra o *matching* do APSARA maximizado, melhorando consideravelmente o desempenho do algoritmo.

3.3 ILQF

O algoritmo LQF (*Longest Queue First*) (MCKEOWN et al., 1996), considera a ocupação da fila, atribuindo um peso $w_{i,j}(n) = L_{i,j}(n)$, onde $L_{i,j}(n)$ representa a dimensão (q_{ij}) de cada VOQ. Filas com ocupações maiores terão atribuições de pesos maiores, e serão portanto, as mais propensas a serem atendidas. O LQF resulta em 100% de *throughput*, no entanto, pode

não atender permanentemente uma fila não vazia.

Considere um comutador de 2×2 portas com $L_{i,j}(0) = 1$ para todo i, j e $\lambda_{1,1} = 1$. No primeiro *time slot*, uma chegada ocorrerá em $Q_{1,1}$ e assim $Q_{1,2}$ continuará não atendida. Logo, durante chegadas contínuas para $Q_{1,1}$, $Q_{1,2}$ permanecerá indefinidamente não atendida.

Teorema 1: O algoritmo de correspondência de peso máximo iLQF é estável (garante a transferência de 100% dos pacotes) para todo processo de chegada admissível.

O escalonamento de um comutador $N \times N$, realizado pelo algoritmo iLQF, é representado pela equação:

$$E[L^T(n+1)L(n+1) - L^T(n)L(n)|L(n)] \leq -\varepsilon\|L(n)\| + k \tag{2}$$

onde $k > 0, \varepsilon > 0$.

$V(n) = L^T(n)L(n)$ é uma função de Lyapunov de segunda ordem e, utiliza o resultado de (KUMAR; MEYN, 1995), que monstra a estabilidade do sistema. O termo $-\varepsilon\|L(n)\|$, indica que se a ocupação das filas de entrada for suficientemente grande, o desvio esperado será negativo.

Em trabalhos anteriores (MCKEOWN et al., 1996) e (MEKKITTIKUL; MCKEOWN, 1996), constatou-se que o algoritmo iLQF pode atingir 100% de *througput*, tanto para o tráfego uniforme como para o tráfego não uniforme, ao considerar as ocupações de filas. Esse algoritmo provê atendimento preferencial a filas grandes, usando um algoritmo de peso máximo, onde cada peso é definido como a ocupação da fila correspondente. Mas é muito difícil de implementá-lo em hardware de alta velocidade, pois apresenta uma complexidade em tempo de execução de $O(N^3 logN)$. Ainda, requer um grande número de comparadores de múltiplos bits para executar várias análises de pesos em paralelo. E portanto, inadequado para comutadores que operam em alta velocidade.

3.4 RPA

O algoritmo RPA (*Reservation with Preemption and Acknowledgment*) foi proposto em (AJMONE MARSAN et al., 1999b). O nome indica que o algoritmo de escalonamento é baseado em uma *rodada de reserva*, onde as portas de entrada do comutador podem indicar a necessidade de transferência das células mais urgentes, possivelmente substituindo solicitações menos urgentes de outras portas de entrada, e uma *rodada de reconhecimento* permitindo que as portas de entrada determinem as células que realmente deverão ser transmitidas às portas de

saída.

Este algoritmo, como todos os outros, requer em cada porta de entrada a disponibilidade de N filas armazenando células direcionadas a diferentes portas de saída. Atua em uma matriz, onde as reservas são recalculadas através de solicitações das portas de entrada a cada *time slot*. Esta matriz de reservas é representada por RES, onde $RES(j)$ se refere a porta de saída j.

Cada elemento $RES(j)$ compreende três campos:

- $RES(j).port.id$: contém a identificação da porta de entrada tentando reservar uma célula para transmitir à porta de saída j.
- $RES(j).urg$: contém a urgência (um valor maior que zero) de transferência da célula de $RES(j).port.id$ à porta de saída j. A urgência define a importância de transferência da célula.
- $RES(j).busy$: é definido como 1 durante a rodada de reconhecimento de uma porta de entrada, indicando a transferência de uma célula à porta de saída j.

Os três campos são inicializados com valores nulos no início de cada rodada reserva.

As portas de entrada acessam a matriz RES seguindo uma ordem pré-estabelecida. Para a descrição destas operações, suponha-se que, em algum *time slot* arbitrário, i é selecionada como a primeira porta de entrada em ordem de acesso; a próxima porta de entrada segue, por exemplo, ascendentemente a ordem subscrita.

Na rodada de reserva, a porta de entrada i seleciona a célula com maior urgência, reservando a porta de saída para a transferência desta célula. Então o índice da porta i é gravado no campo $RES(j).port.id$, e a urgência da célula selecionada é gravada em $RES(j).urg$.

A avaliação do peso é dada pela equação 3 para cada porta de saída:

$$W(j) = U(Q_j) - RES(j).urg \tag{3}$$

onde, $U(Q_j)$ é a urgência da primeira célula da fila Q_j.

Então a porta de entrada i calcula $W(j_{max}) = max_j W(j)$ e grava o valor $x = j_{max}$ quando a função atinge seu máximo. Se $W(x) \geq 0$, a porta i permite a reserva de transferência da célula correspondente, gravando seu índice em $RES(j).port.id$ e $U(Q_x)$ em $RES(j).urg$. Se

$W(j_{max}) \leq 0$, nenhuma reserva pode ser criada. A rodada de reserva continua até que todas as portas sejam processadas, após isso a rodada de reserva é terminada.

Neste ponto, a rodada de reconhecimento é imediatamente iniciada, e a matriz RES é processada pela segunda vez, em todas as portas de entrada, seguindo a mesma ordem utilizada na rodada de reserva. Então verifica as reservas podendo sobrescrever o índice de uma determinada porta de entrada por outra que possua uma célula com maior urgência de transferência. Se a reserva não for sobrescrita, a transferência é concedida e o campo $RES(j).busy$ é alterado para 1. Após o término da execução do algoritmo de reconhecimento, uma célula de cada porta de entrada é transferida para cada porta de saída selecionada.

Logo, o algoritmo de escalonamento RPA apresenta, devido à sua implementação e resultados apresentados em (AJMONE MARSAN et al., 1999b), uma exploração justa e eficiente sob diversas condições críticas de tráfego. Porém, torna-se impraticável pois apresenta uma complexidade computacional de $O(N^2)$.

3.5 ISLIP

O escalonador iSLIP (MCKEOWN, 1999) é uma variação do algoritmo de escalonamento RRM (*Round Robin Matching*). O algoritmo RRM é implementado com codificação de prioridade, onde utiliza o algoritmo *round-robin*, que é simples e pode ser executado de forma muito rápida. Este algoritmo tenta convergir rapidamente através de três etapas:

- *Request*: Cada entrada envia uma requisição a todas as saídas, as quais possuem células a ser transferidas.
- *Grant*: Se uma saída recebe várias requisições de portas de entrada, escolhe uma dentre elas através do escalonador *round-robin*, iniciando pelo elemento de maior prioridade. Então a saída notifica as entradas sobre sua escolha.
- *Accept*: Se uma entrada é escolhida por mais de uma saída, essa entrada escolhe uma saída, através do escalonador *round-robin*, iniciando pela correspondência de maior prioridade.

O algoritmo iSLIP melhora as características do RRM, deixando de reservar e mover pontos de concessão no início do processo de decisão, e só o fará quando a correspondência tenha sido realmente aceita. O iSLIP é idêntico ao RRM com exceção de uma condição no

momento de atualização dos pontos de concessão. A etapa de concessão (*Grant*) de RRM é alterada para:

- *Grant*: Se uma saída recebe várias requisições, escolhe uma dentre elas através do escalonador *round-robin*, iniciando pelo elemento de maior prioridade. Então a saída notifica cada entrada se sua requisição foi ou não concedida. E a concessão para uma entrada é realmente concedida, se e somente se, a concessão for aceita na terceira etapa (*Accept*).

A complexidade de implementação desse algoritmo independe do número de iterações. Quando iterações múltiplas são utilizadas, o número de descisões permanece inalterado, então, o controle de *overhead* necessário para iterações múltiplas também é pequeno.

3.6 DESEMPENHO DOS ALGORITMOS DE ESCALONAMENTO

Os desempenhos dos algoritmos de escalonamento discutidos nesse capítulo são apresentados na Figura 7. As curvas mostram o tamanho médio das filas de entrada ($E[L]$), onde o iLQF, o RPA e o iSLIP apresentam resultados relevantes, porém com desempenho baixo, principalmente em altas cargas.

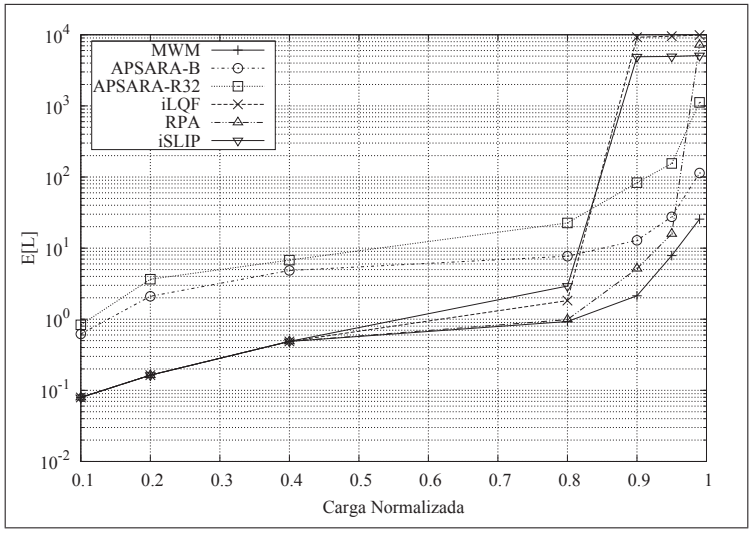

Figura 7: *Tamanho médio das filas para vários escalonadores sob tráfego diagonal*
Fonte: (GIACCONE, 2002)

O algoritmo MWM, mesmo com sua complexidade de implementação muito alta, apresenta resultados ótimos para todos os casos. O algoritmo APSARA apresenta resultados

relevantes mesmo em altas cargas e ainda possui características similares ao algoritmo proposto nessa dissertação. Portanto, todos os testes de desempenho do algoritmo GENIUS são comparados ao MWM e ao APSARA.

3.7 ESCALONADOR BASEADO EM AG SIMILAR AO PROPOSTO

A proposta de (RAGHPATHIKUMAR; RAJA, 2011), foi a construção de um algoritmo genético baseado em um escalonamento para comutadores com filas de entrada. O estudo utiliza codificação de vetores binários como cromossomos. A população inicial deste algoritmo é obtida a partir de um método de permutação.

3.7.1 FUNCÃO DE FITNESS

A função de fitness define a qualidade de um cormossomo e sua sobrevivência na próxima geração, ou seja, neste caso um alto valor de fitness para um determinado cromossomo eleva a probabilidade de subrevivência. A função de fitness é obtida através da equação 4.

$$F = \sum_{i=0}^{N} q_{ij}\lambda_{ij}, j = 1...N \tag{4}$$

3.7.2 SELEÇÃO

Neste estudo foi utilizado a estratégia de seleção por torneio, considerando o torneio de tamanho 2. O indivíduo com mais aptidão (o vencedor) é um dos utilizados como pai para a reprodução.

3.7.3 CROSSOVER

O crossover explora rapidamente o espaço de busca e a cada operação de crossover realizada são criados dois novos indivíduos. A operação de crossover é aplicada com um único ponto de corte definido de forma randômica e é aplicada uma probabilidade p_c igual a $0,5$.

3.7.4 MUTAÇÃO

A mutação fornece uma busca aleatória e pode a garantir que nenhum nenhum espaço de busca deixe de ser examinado. Utiliza uma probabilidade p_m de ocorrência baixa, igual a $0,001$.

3.8 PARÂMETROS UTILIZADOS

As tabelas 1 e 2 definem os parâmetros do algoritmo genético e a configuração do comutador respectivamente simulado por (RAGHPATHIKUMAR; RAJA, 2011).

Tabela 1: Parâmetros do algortimo genético.

Parâmetros	Valores
Tamanho do cromossomo	16
Tamanho da população	32
Probabilidade de crossover	0,5
Probabilidade de mutação	0,001

Fonte: (RAGHPATHIKUMAR; RAJA, 2011)

Tabela 2: Parâmetros do comutador com filas de entrada.

Parâmetros	Valores
Portas de entrada	4
Portas de saída	4
Tamanho do comutador	16
Capacidade filas virtuais (VOQ)	10.000

Fonte: (RAGHPATHIKUMAR; RAJA, 2011)

3.9 RESULTADOS DE SIMULAÇÃO

O estudo de (RAGHPATHIKUMAR; RAJA, 2011) ao investigar a aplicação de algoritmos genéticos no escalonamento de comutadores com filas de entrada, mostra que é possível obter soluções viáveis para o problema. Através dos resultados de simulação o algoritmo apresentou um *throughput* aceitável para as configurações do comutador e parâmentros utilizados.

4 ALGORITMOS GENÉTICOS

Os algoritmos genéticos (AGs) são heurísticas de otimização estocásticas, onde explorações no espaço de soluções são conduzidas imitando a genética de população estabelecida na teoria da evolução de Darwin. Os princípios básicos dos AGs foram estabelecidos de forma detalhada por Holland (HOLLAND, 1975). Para utilizar um algoritmo genético, é necessário representar uma solução para um problema como um genoma (ou *cromossomo*). Os operadores genéticos trabalham sobre aspectos matemáticos abstratos como uma sequência numérica (por exemplo, os cromossomos), o espaço genótipo. *Seleção*, *operadores genéticos* e *substituição*, diretamente derivados dos mecanismos de evolução natural são aplicados a uma população de soluções, favorecem o nascimento e sobrevivência das melhores soluções. A descrição em pseudo-código de um AG é mostrada na Figura 8.

A população compreende um grupo de N_I indivíduos (cromossomos), dos quais candidatos podem ser selecionados para a solução de um problema. Inicialmente, uma população é gerada aleatoriamente. Os valores de *fitness* de todos os cromossomos são avaliados através do cálculo da função objetivo em forma decodificada (fenótipo). Um grupo particular de cromossomos (pais) é selecionado a partir da população para gerar os filhos através das operações genéticas *cruzamento* e *mutação*. A aptidão dos filhos é avaliada de forma semelhante a de seus pais. Os indivíduos da população corrente são substituídos pelos seus descendentes, com base em uma estratégia de substituição. Tal ciclo do AG é repetido até que um critério de parada seja atingido (por exemplo, um número predefinido de gerações N_G é alcançado). Assim, ao longo deste processo de evolução simulada, o melhor cromossomo na população final pode se tornar uma solução altamente evoluída para o problema.

4.1 APLICABILIDADE

Algoritmos genéticos são aplicados em problemas complexos de otimização, problemas com diversos parâmetros ou características que precisam ser combinadas com o intuito de encontrar as melhores soluções. Também em problemas com muitas restrições ou

```
PROCEDIMENTO AG();
(1)  INICIO
(2)     Insere parâmetros do projeto;
(3)     Inicializa população aleatoriamente;
(4)     Calcula o fitness de todos os indivíduos;
(5)     ENQUANTO critério de parada não é atendido FAÇA
(6)        Aplica seleção de pais;
(7)        Aplica crossover com probabilidade  p_c;
(8)        Aplica mutação com probabilidade  p_m;
(9)        Calcula fitness dos indivíduos reproduzidos;
(10)       Aplica estratégia de substituição;
(11)    FIM ENQUANTO
(12)  RETORNE melhor indivíduo;
(13)  FIM;
```

Figura 8: Pseudocodigo algoritmo genético.

condições que não podem ser representadas matematicamente e problemas com grandes espaços de busca. Têm sido aplicados a diversos problemas de otimização (MICHALEWICZ, 1994), tais como: Otimização de Funções Matemáticas, Otimização Combinatória, Otimização de Planejamento, Otimização de *Layout* de Circuitos, Otimização de Distribuição, Otimização em Negócios e Síntese de Circuitos Eletrônicos.

4.1.1 REPRESENTAÇÃO

As soluções do espaço de busca de um problema precisam ser representadas de alguma forma, portanto a representação define a estrutura do cromossomo a ser alterado pelo algoritmo. A representação do cromossomo depende especialmente do tipo do problema proposto e ainda o que se procura manipular geneticamente.

4.1.2 DECODIFICAÇÃO

A decodificação do cromossomo consiste basicamente na construção da solução real do problema a partir do cromossomo. O processo de decodificação constrói a solução para que esta seja avaliada pelo sistema, a fim de possibilitar a classificação das soluções encontradas.

4.1.3 AVALIAÇÃO

A avaliação é a única ligação entre o AG e o problema real. A avaliação é feita através de uma função que melhor representa o problema e tem por objetivo fornecer uma medida de aptidão (*fitness*) de cada solução (indivíduo na população) corrente, isso irá direcionar o

processo de busca. É importante ressaltar que as funções de avaliação são específicas para cada problema.

4.2 SELEÇÃO

O processo de seleção em algoritmos genéticos escolhe indivíduos para a reprodução. A seleção tipicamente é baseada na aptidão dos indivíduos: indivíduos mais aptos têm maior probabilidade de serem escolhidos para reprodução.

Esta seleção normalmente é implementada por uma roleta onde cada indivíduo é representado de forma proporcional a sua aptidão relativa. O operador de seleção é um componente essencial em algoritmos genéticos. Pode-se identificar cinco principais mecanismos de seleção: proporcional, por torneios, com truncamento, por normalização linear e por normalização exponencial (BLICKLE, 1996)(HOLLAND, 1975).

Um mecanismo de seleção é definido pela pressão seletiva ou intensidade de seleção que o mesmo introduz no algoritmo genético. O termo pressão seletiva é utilizado em diferentes contextos e com significados diferentes na literatura de computação evolucionária. A definição de intensidade de seleção empregada em genética é a variação na aptidão média da população induzida pelo método de seleção (BLICKLE, 1996).

Então o principal intuito da seleção é escolher os elementos da população que devem se reproduzir, essa prática proporciona maior chance de reprodução aos membros da população mais adaptados ao problema, isto é, àqueles que apresentam melhor *fitness* (DAVIS, 1996)(CANTONI et al., 2000).

4.3 OPERADORES GENÉTICOS

O objetivo dos operadores de crossover e mutação é fazer com que os cromossomos criados durante o processo de reprodução sejam diferentes dos cromossomos dos pais porém mantendo alguma característica. O crossover é responsável por combinar os cromossomos dos pais na criação dos cromossomos filhos, e o operador de mutação é responsável pela introdução de pequenas mudanças aleatórias nos cromossomos reproduzidos (filhos). Alguns parâmetros influem no comportamento dos Algoritmos Genéticos, sendo estabelecidos conforme as necessidades do problema e dos recursos disponíveis.

4.4 PARÂMETROS DO AG

O tamanho da população determina o número de cromossomos na população, afetando o desempenho global e a eficiência dos AGs. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema e possibilita uma diversidade genética das possíveis soluções. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.

A probabilidade de crossover (p_c) determina a ocorrência do *crossover*. Quanto maior for esta probabilidade, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, dependendo da técnica de substituição, a maior parte da população será substituída, e pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

A probabilidade de mutação (p_m) determina a ocorrência da mutação. Uma baixa probabilidade de mutação previne que uma dada posição fique estagnada em um valor, causando uma convergência prematura, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Com uma probabilidade muito alta a busca se torna essencialmente aleatória.

Os parâmetros genéticos acima citados afetam diretamente o desempenho dos AGs. Os efeitos decorrentes da escolha inadequada destes parâmetros vão desde aumento no tempo de convergência, convergência prematura, estagnação da busca, maior necessidade de recursos computacionais e até a não convergência para uma solução viável.

4.5 SUBSTITUIÇÃO

As técnicas de substituição podem ser classificadas em *generational* e *steady state*. A maioria dos trabalhos tem utilizado a estratégia *generational*, onde toda a população é substituída a cada geração. No entanto, uma tendência mais recente tem favorecido a substituição do tipo *steady state*. Essa operação funciona com outro extremo, onde em cada geração apenas alguns (normalmente dois) indivíduos são substituídos.

Em alguns casos este pode ser um modelo melhor do que o que acontece na natureza. Nas espécies de vida curta, incluindo alguns insetos, os pais põem ovos, e depois morrem até mesmo antes do nascimento de seus filhos. Mas, em espécies de vida mais longa, incluindo mamíferos, pais e filhos podem estar vivos ao mesmo tempo. Isso permite que os pais possam

cuidar e ensinar seus filhos, mas também dá origem a concorrência entre eles (BEASLEY et al., 1993).

No caso da substituição *steady state*, não só deve-se considerar como escolher dois indivíduos para serem pais, mas também selecionar dois indivíduos da população que serão descartados, liberando espaço na população para acolher novos indivíduos. Vários esquemas são possíveis (BEASLEY et al., 1993), incluindo:

- Seleção de pais de acordo com a aptidão e seleção de substituições ao acaso;
- Seleção de pais de forma aleatória, e seleção de substituições por aptidão inversa;
- Seleção de ambos os pais e as substituições de acordo com a aptidão ou aptidão inversa.

A diferença essencial entre, uma substituição *generational* e uma substituição *steady state*, é que as estatísticas de população (tais como aptidão) são recalculadas após cada reprodução e na substituição *steady state* (isso não precisa ser computacionalmente caro se realizado de de forma incremental), com isso os novos indivíduos estarão imediatamente à disposição para reprodução. Essa característica é importante em AG pois gera a oportunidade de explorar um indivíduo promissor assim que ele é criado (BEASLEY et al., 1993).

O Elitismo é o método mais utilizado para melhorar a convergência dos AGs e é uma adição aos métodos de seleção que força os AGs a reter um certo número de melhores indivíduos em cada geração (MILLER; GOLDBERG, 1995). Tais indivíduos podem ser perdidos se eles não forem selecionados para reprodução ou se eles forem danificados pelos operadores genéticos.

5 O ESCALONADOR GENIUS

Para a utilização de um algoritmo genético, é necessário representar uma solução para um problema como um *cromossomo*. *Seleção*, *operadores genéticos* e *substituição*, diretamente derivados dos mecanismos de evolução natural são, então, aplicados a uma população de soluções favorecendo o nascimento e sobrevivência das melhores soluções.

Nessa dissertação, as características naturais de busca em ambientes dinâmicos dos AGs são exploradas na construção de um algoritmo de escalonamento de ótimo desempenho denominado GENIUS (*Genetic Scheduling Algorithm for High-Performance Switches*). A descrição em pseudo-código do escalonador proposto é mostrada na Figura 9.

De acordo com características apresentadas na Seção 3.2, levantadas por Giaccone et al (GIACCONE et al., 2003), sugerem que as ocupações médias das filas mudam muito pouco durante uma sequência de *time slots*. Portanto, é provável que *matchings* pesados em um determinado *time slot*, continuarão pesados por mais alguns *time slots*. Essas caracteríscas garantem a convergência do escalonador genético para soluções ótimas.

A população compreende um grupo de K indivíduos (cromossomos), dos quais candidatos podem ser selecionados para a solução de um problema. Inicialmente, uma população é gerada aleatoriamente. Os valores de *fitness* de todos os cromossomos são avaliados através do cálculo da função objetivo. Um grupo particular de cromossomos (pais) é selecionado à partir da população para gerar os filhos através das operações genéticas *crossover* e *mutação*. A aptidão dos filhos é avaliada de forma semelhante a de seus pais. Os indivíduos da população corrente são substituídos pelos seus descendentes, com base em uma estratégia de substituição. A evolução da população avança ao longo da execução do escalonador, e a cada *time slot* o melhor indivíduo da população corresponde ao *matching* a ser utilizado pelo comutador.

```
PROCEDIMENTO Escalonador Genético( );
(1)  INICIO
(2)     Insere parâmetros do sistema;
(3)     Inicializa população aleatoriamente;
(4)     Avalia o fitness de cada indivíduo;
(5)     PARA cada time slot FAÇA
(6)         Aplica seleção de pais;
(7)         Aplica crossover com probabilidade p_c;
(8)         Aplica mutação com probabilidade p_m;
(9)         Avalia o fitness de todos os indivíduos;
(10)        Atualiza população;
(11)        matching = melhor indivíduo;
(12)    FIM PARA
(13) FIM
```

Figura 9: Pseudocódigo do algoritmo GENIUS.

5.1 CODIFICAÇÃO E OPERADORES GENÉTICOS

Os detalhes específicos associados ao GENIUS são apresentados nos parágrafos seguintes.

Esquema de codificação: Nesse trabalho, um *matching* é um vetor de inteiros, de tamanho *N*, usado para representar a correspondência de portas de entrada com portas de saída. Assim, um cromossomo corresponde, através dos índices, à porta de entrada e, através do conteúdo, à uma sequência de *N* portas de saída, como na Figura 10.

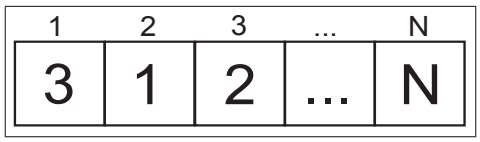

Figura 10: *Representação do cromossomo.*

Avaliação do fitness: O fitness de cromossomo corresponde simplesmente ao peso do matching definido na Capítulo 3.

Seleção de pais: A seleção dos pais emula o mecanismo natural de "sobrevivência do mais apto". Neste trabalho foi utilizado o método de seleção torneio de tamanho *k*, onde *k* indivíduos são escolhidos aleatoriamente e aquele com o maior fitness (o que "vence o torneio") é usado como um pai. A seleção torneio é então repetida com uma nova seleção de *k* indivíduos a fim de encontrar outro pai de características diferentes (GOLDBERG et al., 1993; MILLER; GOLDBERG, 1995).

Operadores genéticos: O *crossover* é um operador de recombinação usado para produzir filhos. Neste trabalho é utilizado o crossover OX *(Order Crossover)* de dois pontos (GOLDBARG; LUNA, 2000). Dado dois pais, a partir do segundo ponto de corte de um dos pais, descreve-se a sequência e posteriormente remove-se, desta, os genes existentes entre os cortes, ao final os filhos herdam as sub-sequências entre os pontos de corte. Estes pontos são selecionados aleatoriamente. O processo é exemplificado na Figura 11.

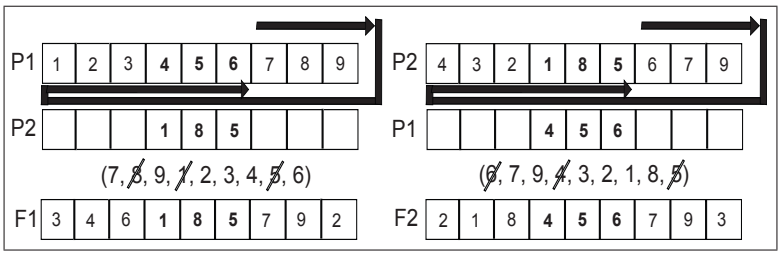

Figura 11: *Crossover OX (Order Crossover).*

A *mutação* é usada para evitar a convergência das soluções para ótimos locais de baixa qualidade. Nesta proposta foram obtidos bons resultados usando o operador de mutação SIM (*Simple Inversion Mutation*) que faz seleção aleatória de pontos de mutação, e inverte os conjuntos de genes entre estes pontos para cada descendente produzido (HOLLAND, 1975)(GREFENSTETTE, 1987). Considerando apenas um ponto de mutação a Figura 12 exemplifica a operação.

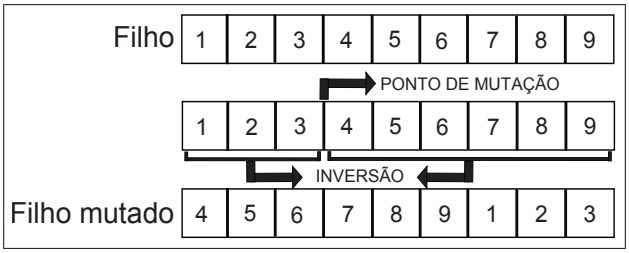

Figura 12: *Mutação SIM (Simple Inversion Mutation).*

Estratégia de substituição 1: A fim de gerar uma nova população foi utilizada a *estratégia elitista* onde uma vez que a população de filhos é gerada, ela é mesclada com a população de pais segundo a seguinte regra: somente os melhores indivíduos presentes nas populações de pais e filhos entram na nova população. Para isso foi utilizado o algoritmo de classificação *heapsort*, que em seu pior caso apresenta uma complexidade computacional $O(K\ logK)$ (CORMEN et al., 2003). Essa estratégia é aplicada na versão elitista (GENIUS-E) do escalonador proposto.

Estratégia de substituição 2: A fim de reduzir a complexidade do algoritmo de escalonamento, a geração da nova população é através da comparação imediata realizada entre

filhos e pais, e segue a seguinte regra: os melhores indivíduos presentes após a reprodução entre pais e filhos entram na nova população. Como não há qualquer classificação nesta estratégia a complexidade computacional exigida é $O(K)$. Essa estratégia é representada pela Figura 13 e é aplicada na versão simplificada (GENIUS-S) do escalonador proposto.

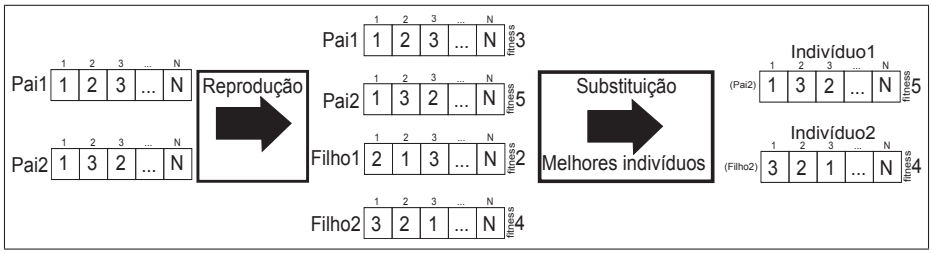

Figura 13: *Representação da estratégia de substituição 2.*

Tamanho da população: O tamanho da população K foi definido de forma empírica de modo a não elevar demasiadamente a complexidade do escalonador.

5.2 VERSÃO MAX DO ESCALONADOR GENIUS

Uma versão MAX também foi implementada, seguindo a lógica semelhante ao APSARA. Nesse caso, após a geração da nova população o melhor indivíduo é submetido ao *submatching*, criando uma nova tabela de pesos a partir das correspondências com pesos nulos e aplicando o algoritmo de aproximação simples de correspondência ponderada (PREIS, 1999).

5.3 IMPLEMENTAÇÃO

Uma implementação "tradicional" do escalonador GENIUS levaria a uma complexidade temporal elevada, em função da característica serial dos Algoritmos Genéticos. Uma solução para este problema corresponderia a fazer o uso do conceito de Algoritmo Genético em "pipe-line" (PAKHIRA; DE, 2007) onde K operações genéticas são realizadas ao mesmo tempo sobre uma população de K indivíduos produzindo K filhos. A operação de substituição é realizada após esta etapa para a criação da nova população. Desta forma, o GENIUS passaria a apresentar uma complexidade espacial $O(K)$ e complexidade temporal descrita pela Tabela 3.

Tabela 3: Complexidades das versões do algoritmo GENIUS.

Versão	$K = \binom{N}{2}$	$K = N$
GENIUS-S	$O(N)$	$O(N)$
GENIUS-E	$O(N^2\ logN)$	$O(N\ logN)$

6 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados resultados numéricos obtidos via simulação (com base no método *batch means* (BANKS, 1998) (ALEXOPOULOS et al., 1997)), considerando-se os algoritmos: GENIUS, APSARA e MWM.

6.1 CONFIGURAÇÕES DA SIMULAÇÃO

Comutador: Foi considerado um comutador com $N = 32$ portas. Cada VOQ pode armazenar até 10.000 pacotes, sendo que, pacotes em excesso são descartados.

Tráfego de entrada: A carga para todas as entradas foi normalizada, sendo $\rho \in (0,1)$ seu valor. O processo de chegada é independente e identicamente distribuído seguindo um processo de Bernoulli. Foram utilizadas as seguintes matrizes de carga, onde $|i| = (i \; mod \; N)$, para testar o desempenho dos algoritmos:

- *Tráfego uniforme*: Neste caso $\lambda_{ij} = \rho/N \; \forall i, j$. Este é o teste de tráfego mais comumente usado na literatura.
- *Tráfego diagonal*: Neste caso $\lambda_{ii} = 2\rho/3N$, $\lambda_{i|i+1|} = \rho/3N \; \forall i$, and $\lambda_{ij} = 0$ para qualquer outro i e j. Esta é uma carga desbalanceada no sentido de que a entrada i possui pacotes para as saídas i e $|i+1|$ (AJMONE MARSAN et al., 1999a). É mais difícil tratar esse tráfego do que o tráfego uniforme.

Métricas de desempenho: Foram comparados os desempenhos dos diferentes algoritmos, medindo os tamanhos médios das filas de entrada ($E[L]$) e os atrasos médios das filas de entrada ($E[T]$), que foram computados diretamente usando a fórmula de Little.

Para a obtenção dos resultados numéricos foi desenvolvido um software específico baseado em simulação de eventos discretos (HAVERKORT, 1998). Esse software foi validado mediante comparação de resultados considerando o algoritmo MWM presente em (GIACCONE et al., 2001). As simulações foram executadas até que a estimativa do atraso médio atingisse

a largura relativa de intervalo de confiança igual a 10%, com uma probabilidade $\geq$ 0,95. A estimativa da largura do intervalo de confiança foi obtida usando o método *batch means*, uma técnica padrão para avaliar resultados de simulação (BANKS, 1998) (ALEXOPOULOS et al., 1997).

6.2 RESULTADOS NUMÉRICOS

Neste item são apresentados e analisados os resultados obtidos, explorando o espaço de parâmetros dos algoritmos de escalonamento. Foram comparados os desempenhos dos diferentes algoritmos, medindo os tamanhos médios ($E[L]$) e o atraso médio de atendimento ($E[T]$) das filas de entrada. Claramente, quanto menor forem os valores para $E[L]$ e $E[T]$ melhor será o desempenho do sistema. O GENIUS foi configurado da seguinte forma: $p_c = 100\%$, $p_m = 65\%$ e $k = 0,25 \times K$. O tamanho da população (K) foi definido utilizando os valores 496 e 32, especificamente mesmos valores utilizados em (GIACCONE, 2002), para efeitos de comparação.

6.2.1 VARIAÇÃO DA POPULAÇÃO

As Figuras 14 e 15 mostram curvas de desempenho, mais especificamente, o tamanho médio das filas de entrada versus a carga normalizada imposta sobre as portas de entrada, para os tráfegos uniforme e diagonal respectivamente, considerando dois diferentes parâmetros para o tamanho da população. Em primeiro lugar, observa-se que $E[L]$ aumenta com a carga normalizada como o esperado, e permanecem limitados para cargas iguais a 99%. Com o aumento do tamanho da população os resultados mostram também aumento do desempenho para ambos os escalonadores GENIUS e APASARA , como esperado. Além disso, o GENIUS-E mostra um desempenho melhor comparado ao APASARA, considerando $K = 496$, especialmente para o tráfego diagonal.

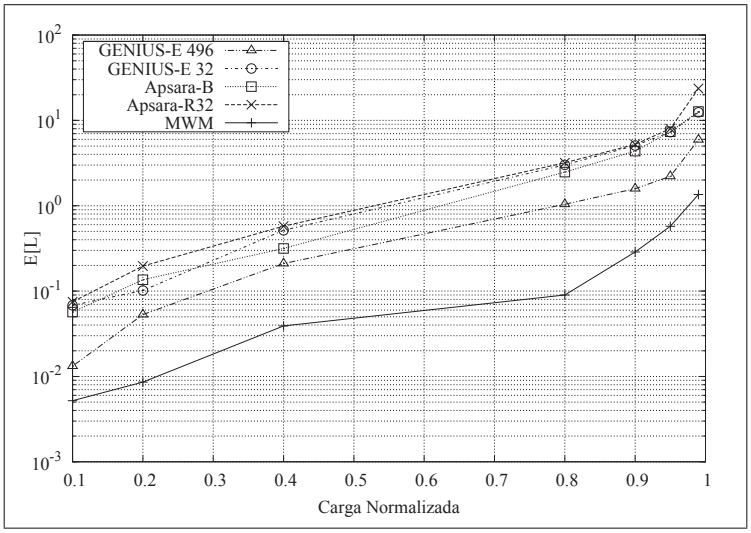

Figura 14: *Tráfego Uniforme – Variação do tamanho da população (GENIUS-E).*

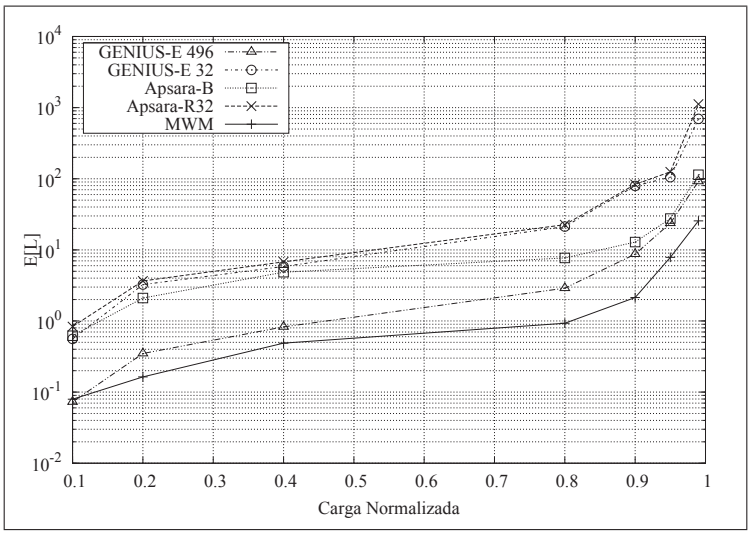

Figura 15: *Tráfego Diagonal – Variação do tamanho da população (GENIUS-E).*

6.2.2 IMPACTO DA REDUÇÃO DE COMPLEXIDADE

As Figuras 16 e 17 apresentam os desempenhos dos escalonadores GENIUS-E e GENIUS-S considerando $K = 496$, para os tráfegos uniforme e diagonal, visando analisar o impacto da complexidade. Nota-se que, ambas as versões do GENIUS apresentam o desempenho semelhante ou melhor, se comparado com o APSARA. Para efeitos de comparação, as Figuras 18 e 19 consideram $K = 32$. Observa-se, então, que o GENIUS-E apresenta o mesmo desempenho do APASARA. Além disso, a redução da complexidade do algoritmo (GENIUS-S), resulta em uma ligeira perda de desempenho.

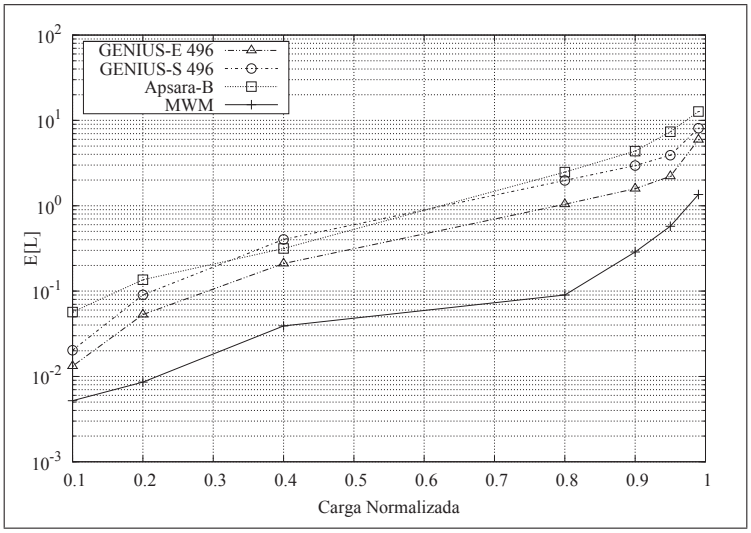

Figura 16: *Tráfego Uniforme – Alteração de complexidade (K=496).*

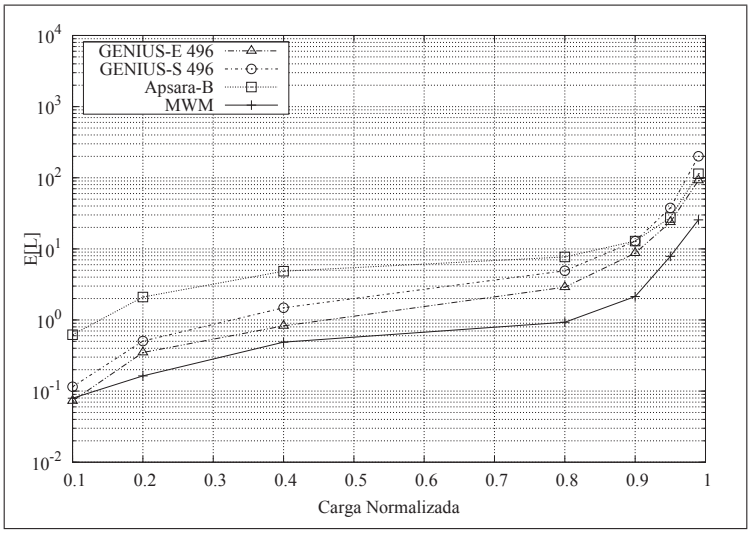

Figura 17: *Tráfego Diagonal – Alteração de complexidade (K=496).*

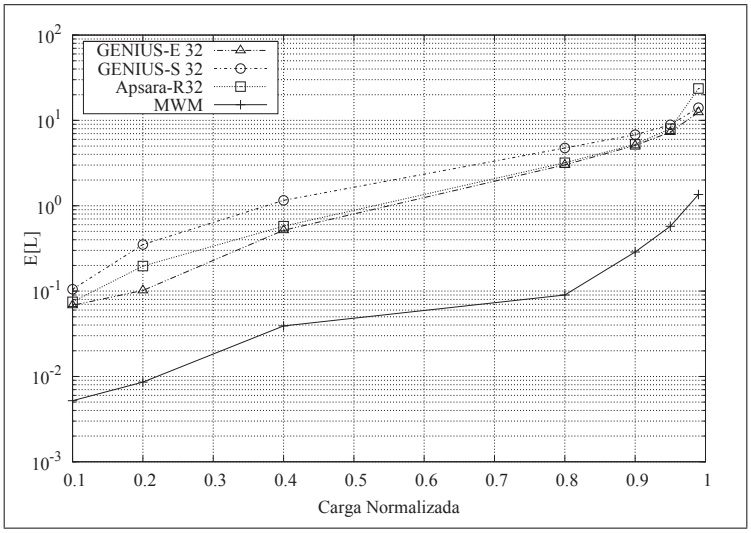

Figura 18: *Tráfego Uniforme – Alteração de complexidade (K=32).*

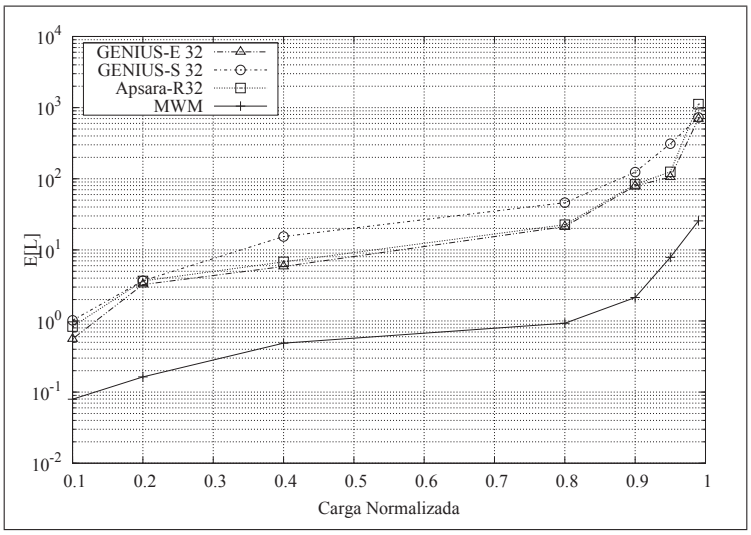

Figura 19: *Tráfego Diagonal – Alteração de complexidade (K=32).*

6.2.3 ANÁLISE DA VERSÃO MAX

Para a versão MAX dos escalonadores GENIUS e APSARA, as Figuras 20 e 21 mostram desempenhos similares (para tamanho da população igual a $K = 496$). Como esperado, todos os algoritmos apresentam as ocupações médias de filas muito próximas do MWM. Porém, com a redução do tamanho da população para $K = 32$ (Figuras 22 e 23) todos os algoritmos desviam do desempenho ótimo (MWM), especialmente em altas cargas. É importante ressaltar que o GENIUS-S e o GENIUS-E têm desempenho semelhante, porém melhor que o desempenho do APSARA.

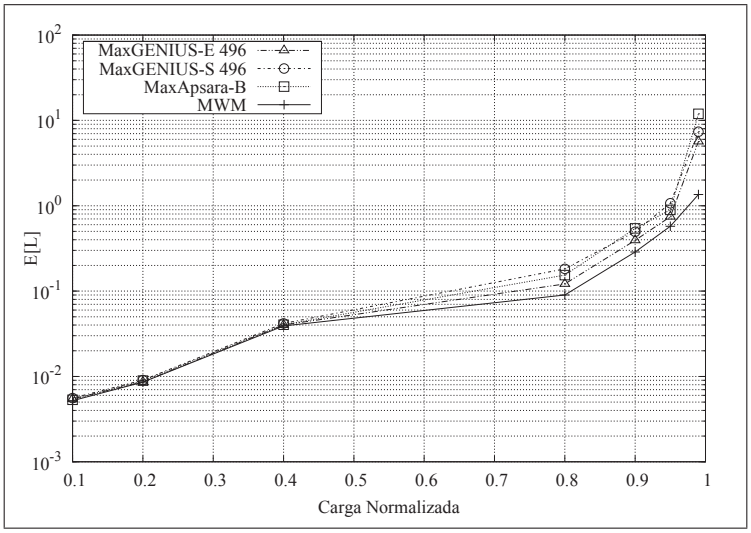

Figura 20: *Tráfego Uniforme – Versão MAX (K=496).*

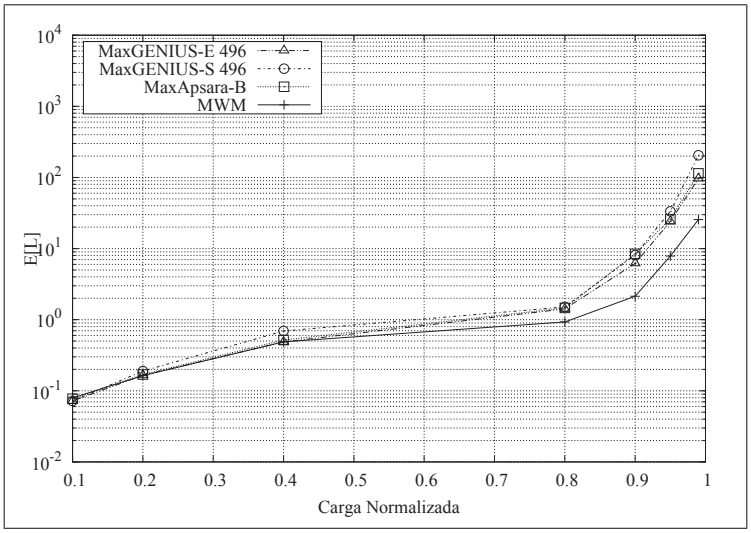

Figura 21: *Tráfego Diagonal – Versão MAX (K=496).*

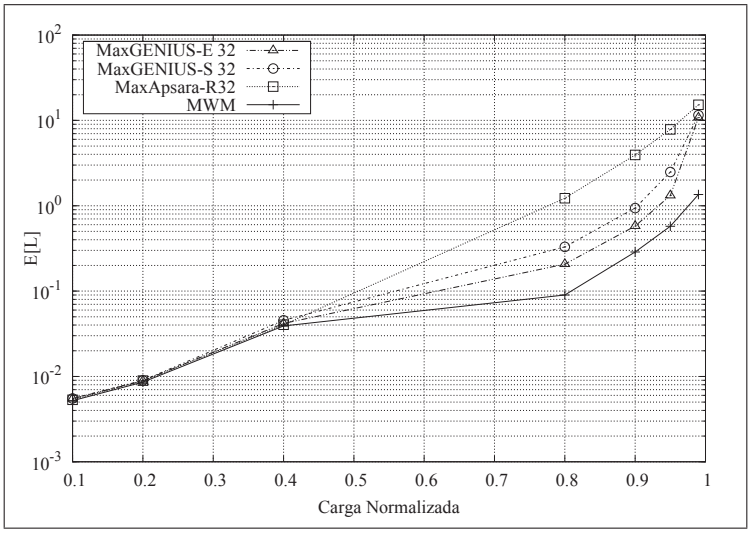

Figura 22: *Tráfego Uniforme – Versão MAX (K=32).*

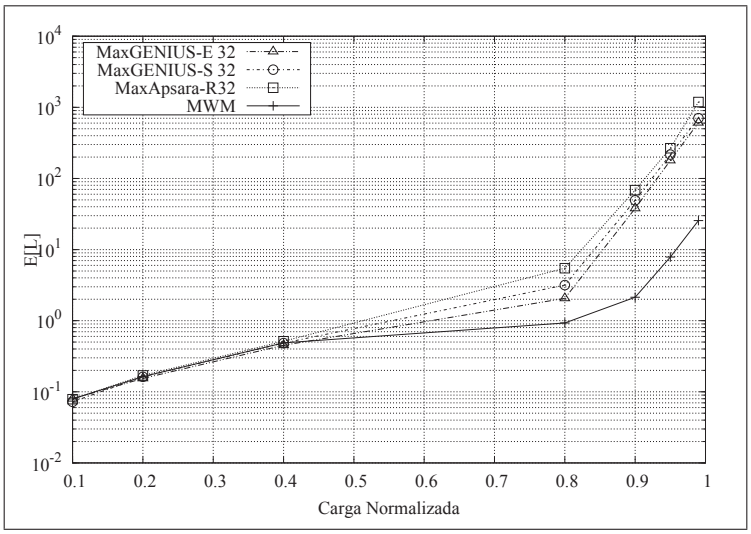

Figura 23: *Tráfego Diagonal – Versão MAX (K=32).*

6.2.4 ATRASO MÉDIO DAS FILAS DE ENTRADA

Os resultados a seguir apresentam curvas de desempenho do atraso médio de atendimento das filas de entrada versus a carga normalizada imposta sobre as portas de entrada, para o tráfego diagonal. A Figura 24 considera dois diferentes parâmetros para o tamanho da população, nela observa-se que atraso médio de atendimento aumenta com a carga normalizada, e permanecem também limitados para cargas iguais a 99%. Como esperado, através de resultados anteriores de ocupação média, com aumento do tamanho da população os resultados apresentam um aumento de desempenho para ambos os escalonadores GENIUS e APASARA. Além disso, o GENIUS-E mostra um desempenho melhor comparado

ao APASARA considerando $K = 496$, especialmente para cargas menores que 99%.

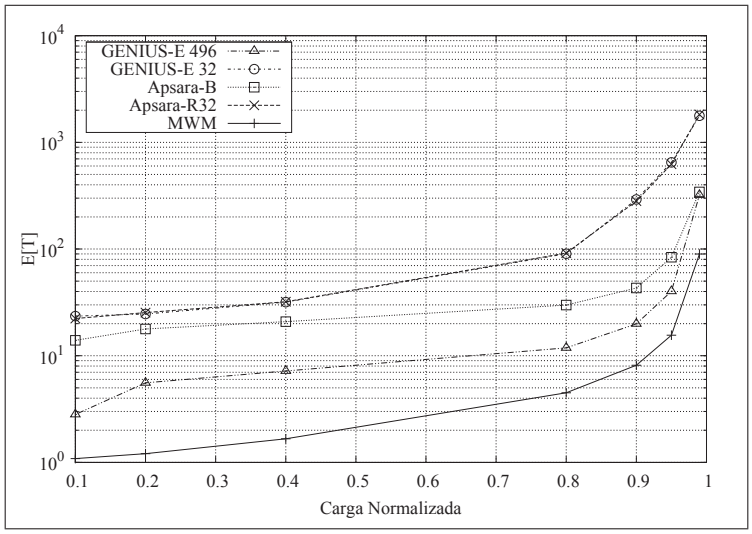

Figura 24: *Tráfego Diagonal – Variação do tamanho da população (GENIUS-E).*

A Figura 25 apresenta os desempenhos dos escalonadores GENIUS-E e GENIUS-S considerando $K = 496$, para o tráfego diagonal. A fim de considerar o impacto da complexidade agora para o atraso médio, nota-se que, ambas as versões do GENIUS apresentam o desempenho semelhante ou melhor comparando com o APSARA. Para efeitos de comparação, a Figura 26 considera $K = 32$. Observa-se que o GENIUS-E apresenta o mesmo desempenho do APASARA. Além disso, reduzindo a complexidade do algoritmo (GENIUS-S), os resultados para o atraso médio continuam resultando numa ligeira perda de desempenho.

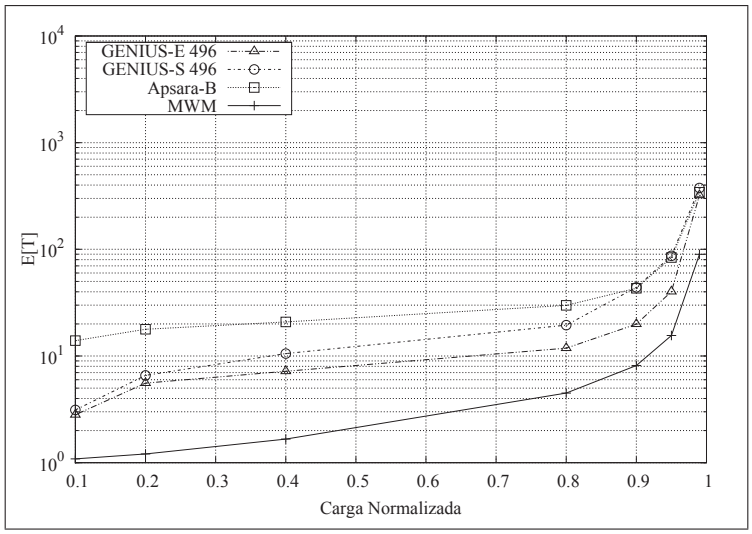

Figura 25: *Tráfego Diagonal – Alteração de complexidade (K=496).*

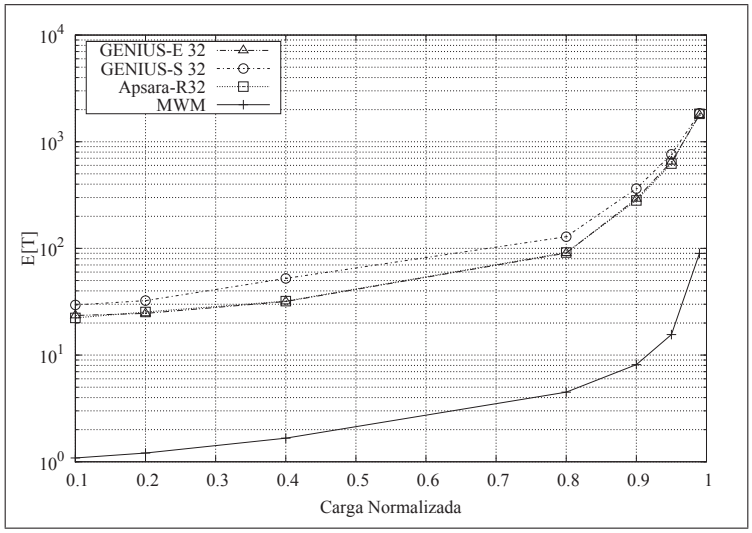

Figura 26: *Tráfego Diagonal – Alteração de complexidade (K=32).*

Nas versões MAX dos escalonadores GENIUS e APSARA, a Figura 27 mostra desempenho similar (para tamanho da população igual a $K = 496$). Como esperado, todos os algoritmos apresentam os atrasos médios de atendimento de filas muito próximos ao MWM. Novamente, com a redução do tamanho da população para $K = 32$ (Figura 28) todos os algoritmos desviam do desempenho ótimo (MWM), especialmente em altas cargas. É importante ressaltar que nessa métrica o GENIUS-S, o GENIUS-E e o APSARA apresentam desempenhos muito semelhantes, confirmando resultados mostrados na Seção 6.2.3.

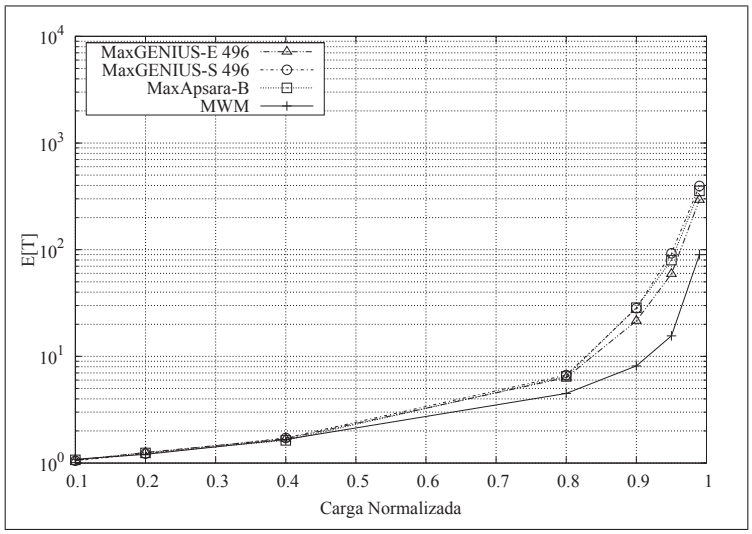

Figura 27: *Tráfego Diagonal – Versão MAX (K=496).*

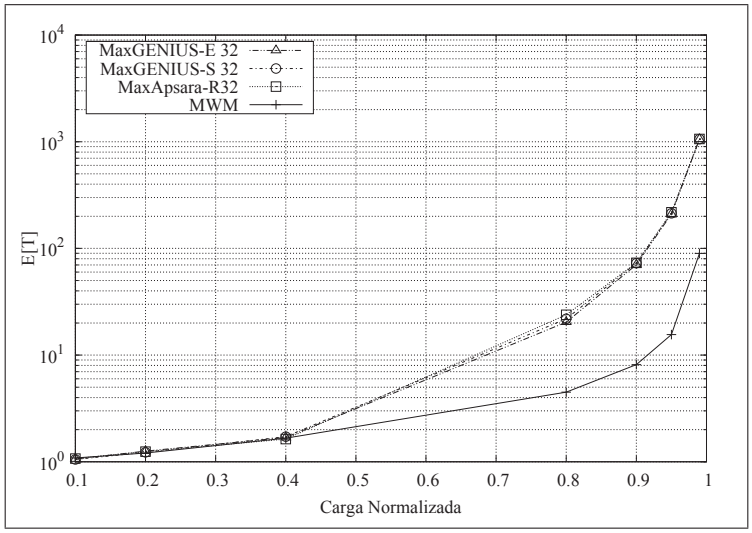

Figura 28: *Tráfego Diagonal – Versão MAX (K=32).*

7 CONCLUSÃO

Esta dissertação apresentou uma abordagem para criação de um escalonador para comutadores com filas de entrada através de algoritmos genéticos. Com a aplicação do escalonador GENIUS foram encontrados bons resultados em relação a outros escalonadores presentes na literatura.

Com complexidades aceitáveis o escalonador GENIUS, através de suas versões elitista (GENIUS-E) e simplificada (GENIUS-S), apresentou desempenhos relevantes, que são semelhantes ou superiores ao escalonador APSARA apresentado por (GIACCONE, 2002) e que se aproximam do desempenho do escalonador ótimo MWM.

A fim de ajustar o sistema, foram testados diferentes parâmetros para o algoritmo genético. Os melhores resultados foram obtidos quando a pressão seletiva (GOLDBERG et al., 1993) foi elevada, com aumento do tamanho do torneio para $k = S \times 0,25$. A cada iteração utilizamos apenas uma única geracão com o intuito de não elevar a complexidade do algoritmo. Entretanto quando elevamos o tamanho do torneio temos um aumento da convergência para um ótimo local.

Outros resultados dessa dissertação podem ser representados pela publicação no XXXI Simpósio Brasileiro de Telecomunicações, do artigo com o título: Escalonamento Baseado em Algoritmos Genéticos para Roteadores de Alto Desempenho (HOFFMANN; WILLE, 2013). E a submissão de um artigo estendido para a revista *Computer Networks* com o título *GENIUS - A Genetic Scheduling Algorithm for High-Performance Switches*.

Em trabalho anterior, Raghpathikumar e Raja (RAGHPATHIKUMAR; RAJA, 2011), mostram uma solução semelhante à presente dissertação quanto a implementação de um escalonador para roteadores com portas de entrada utilizando algoritmos genéticos. Nele, mostram toda a codificação e parâmetros utilizados nos testes. Como método de seleção foi utilizado o método torneio e operadores genéticos: crossover de um ponto com probabilidade de 50% e mutação com probabilidade de 0,1%. Porém os resultados não apresentam as métricas comumente utilizadas ($E[L]$ e $E[T]$), não possibilitando a comparação de desempenho com

outros escalonadores. Apenas mostram que a solução pode apresentar um *throughput* aceitável para um comutador de tamanho 4×4.

7.1 TRABALHOS FUTUROS

O conhecimento produzido a partir desta dissertação representa o ponto de partida para a concepção, avaliação e implementação de um escalonador viável, devido aos resultados obtidos.

Sugere-se como trabalhos futuros o teste com comutadores de maiores dimensões (por exemplo, $N = 64$), e a aplicação de outras técnicas de torneio e operadores genéticos a fim de elevar o desempenho, sem que a complexidade do algoritmo seja elevada.

REFERÊNCIAS

AJMONE MARSAN, M. et al. On the behavior of input queuing switch architectures. **European Transactions on Telecommunications**, 1999.

AJMONE MARSAN, M. et al. Rpa: A flexible scheduling algorithm for input buffered switches. **IEEE Trans. on Communications**, 1999.

ALEXOPOULOS, C.; FISHMAN, G. S.; SEILA, A. F. Computacional experience with the batch means method. **Winter Simulation Conference**, 1997.

BANKS, J. Simulation - principles, methodology, advances, applications, and pratice. **EMP**, 1998.

BEASLEY, D.; BULL, D. R.; MARTIN, R. R. An orverview of genetic algorithms: Part 1, fundamentals. **Inter-University Committee on Computing**, 1993.

BLICKLE, T. **"Theory of Evolucionary Algorithms and Application to System Synthesis"**. Tese (Doutorado) — Federal Institute of Technology, Zurique, 1996.

CANTONI, M.; MARSEGUERRA, M.; ZIO, E. Genetic algorithms and monte carlo simulation for optimal plant design. **Reliab Engng Syst Safety**, 2000.

CORMEN, T. H. et al. Introduction to algorithms (2nd ed.). **MIT Press**, 2003.

DAVIS, L. **Handbook of Genetic Algorithms**. [S.l.]: Stamford: International Thomson Publishing, 1996.

GIACCONE, P. **Queueing and scheduling algorithms for high performance routers**. Tese (Doutorado) — Politecnico di Torino, 2002.

GIACCONE, P.; PRABHAKAR, B.; SHAH, D. Randomized scheduling algorithms for high-aggregate bandwidth switches. **IEEE Journal on Selected Areas in Communications**, 2003.

GIACCONE, P.; SHAH, D.; PRABHAKAR, B. An implementable parallel scheduler for input-queued switches. **IEEE Micro**, 2001.

GOLDBARG, M. C.; LUNA, H. P. L. **Simulating Computer Systems Techniques and Tools**. [S.l.]: Editora Campus, 2000.

GOLDBERG, D. E.; THIERENS, D.; DEB, K. Toward a better understanding of mixing in genetic algorithms. **Journal of the Society of Instrument and Control Engineers**, 1993.

GREFENSTETTE, J. J. Genetic algorithms, tournament selection, and the effects of noise. **Los Altos**, 1987.

GREFENSTETTE, J. J. Genetic algorithms for changing environments. **Parallel Problem Solving from Nature 2**, 1992.

HAVERKORT, B. R. **Performance of Computer Communication Systems: A Model-Based Approach.** [S.l.]: John Wiley & Sons Ltda, 1998.

HOFFMANN, J. R.; WILLE, E. C. G. Escalonamento baseado em algoritmos genéticos para roteadores de alto desempenho. **XXXI Simpósio Brasileiro de Telecomunicações**, 2013.

HOLLAND, J. H. Adaptation in natural and artificial systems. **MIT Press**, 1975.

KAROL, M.; HLUCHYJ, M.; MORGAN, S. Input versus output queuing on space division switch. **IEEE Trans. on Communications**, 1987.

KUHN, H. W. The hungarian method for the assignment problem. **Naval Research Logistics (NRL)**, 2005.

KUMAR, P. R.; MEYN, S. P. Stability of queuing networks and scheduling policies. **IEEE Trans. Automat. Contr.**, 1995.

KUROSE, J. F. **Redes de computadores e a Internet - Uma nova abordagem top-down**. [S.l.]: Pearson Addison Wesley, 2006.

LUGER, G. F. **Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos**. [S.l.]: Bookman, 2004.

MCKEOWN, N. The islip scheduling algorithm for input-queued switches. **IEEE/ACM TRANSACTIONS ON NETWORKING**, 1999.

MCKEOWN, N.; ANANTHARAM, V.; WALRAND, J. Achieving 100% throughput in an input-queued switch. **Proceedings of INFOCOM**, 1996.

MEKKITTIKUL, A.; MCKEOWN, N. A starvation-free algorithm for achieving 100% throughput in an input-queued switch. **Proceedings of ICCCN**, 1996.

MICHALEWICZ, Z. Genetic algorithms+data strutures=evolution programs. **Springer-Verlag**, 1994.

MILLER, B. L.; GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. **IlliGAL Report**, 1995.

PAKHIRA, K.; DE, R. K. Generational pipelined genetic algorithm (plga) using stochastic selection. **Int. Journal of Computer Systems Science and Engineering**, 2007.

PREIS, R. Linear time 1/2 approximation algorithm for maximum weighted matching in general graphs. **Springer - Verlag Berlin Heidelberg**, 1999.

RAGHPATHIKUMAR, D.; RAJA, B. K. A genetic algorithm based scheduling of an input queued switch. **International Journal of Soft Computing**, 2011.

SHAH, D.; GIACCONE, P.; PRABHAKAR, B. Efficient randomized algorithms for input-queued switch scheduling. **IEEE Micro**, 2002.

TASSIULAS, L.; EPHREMIDES, A. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. **IEEE Trans. on Automatic Control**, 1992.

APÊNDICE A – MÉTODO DA MÉDIA DOS LOTES - BATCH MEANS

O método *Batch Means* (BANKS, 1998) (ALEXOPOULOS et al., 1997) é baseado em uma única e longa execução da simulação para a estimação do intervalo de confiança. A sequência original de n observações $(X_1, X_2, ..., X_n)$ é dividida em k lotes não sobrepostos de tamanho m. As médias desses lotes (*batch means*) podem então ser utilizadas para o cálculo do intervalo de confiança. Considerando-se k lotes de m observações cada um, então o *i-ésimo* lote corresponde à sequência:

$$X_{(i-1)m+1}, X_{(i-1)m+2}, ..., X_{im} \tag{5}$$

para $i = 1, 2, ..., k$. A *i-ésima* média do lote é dada por:

$$Y_i(m) = \frac{1}{m} \sum_{j=1}^{m} X_{(i-1)m+j} \tag{6}$$

A média amostral é definida por:

$$\overline{Y}_k = \frac{1}{k} \sum_{i=1}^{k} Y_i(m) \tag{7}$$

Com variância dada por:

$$S_B^2(m,k) = \frac{1}{k-1} \sum_{i=1}^{k} \left(Y_i(m) - \overline{Y}_k\right)^2 \tag{8}$$

Em simulações discretas o problema da correlação é mais complexo de ser tratado. Após o período transiente a simulação entra no período *steady-state* e é neste período que as observações devem ser utilizadas na análise final caso seja optado pela eliminação dos resultados coletados no período transiente.

O objetivo do intervalo de confiança, é estabelecer uma margem de segurança para informações probabilísticas indicando a confiabilidade de uma estimativa. Dessa forma ao

delimitar o intervalo, é possível avaliar matematicamente a diferença entre os resultados obtidos, tendo a probabilidade $(1-\alpha)$ de que a estimativa esteja contida dentro de uma faixa pré estabelecida de valores.

Para o método *Batch Means* o intervalo de confiança a 95% é dado por:

$$\left[\overline{Y}_k - 2,045\frac{S_B(m,k)}{\sqrt{k}}, \overline{Y}_k + 2,045\frac{S_B(m,k)}{\sqrt{k}}\right] \tag{9}$$