UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE ESTATÍSTICA
BACHARELADO EM ESTATÍSTICA

Desenvolvimento e Implementação de Aplicativo Computacional para Otimização em Redes de Filas

Fernando Henrique Rocha da Silva

Ouro Preto-MG
2023

Fernando Henrique Rocha da Silva

Desenvolvimento e Implementação de Aplicativo Computacional para Otimização em Redes de Filas

Monografia de Graduação apresentada ao Departamento de Estatística do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de bacharel em Estatística.

Orientador: Anderson Ribeiro Duarte

Ouro Preto

2023

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S586d Silva, Fernando Henrique Rocha Da.
Desenvolvimento e implementação de aplicativo computacional para otimização em redes de filas. [manuscrito] / Fernando Henrique Rocha Da Silva. - 2023.
57 f.: il.: , tab..

Orientador: Prof. Dr. Anderson Ribeiro Duarte.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Graduação em Estatística .

1. Redes de filas. 2. Otimização. 3. Simulated Annealing. I. Duarte, Anderson Ribeiro. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004

Bibliotecário(a) Responsável: Luciana De Oliveira - SIAPE: 1.937.800

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
COLEGIADO DO CURSO DE ESTATISTICA

FOLHA DE APROVAÇÃO

Fernando Henrique Rocha da Silva

Desenvolvimento e implementação de aplicativo computacional para otimização em redes de filas

Monografia apresentada ao Curso de Estatística da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Estatística

Aprovada em 1º de setembro de 2023

Membros da banca

Dr. Anderson Ribeiro Duarte - Orientador (Universidade Federal de Ouro Preto)
Dr. Helgem de Souza Ribeiro Martins (Universidade Federal de Ouro Preto)
Ms. Gabriel Lima de Souza (Universidade Federal de Ouro Preto)

Professor Dr. Anderson Ribeiro Duarte, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 01/09/2023.

Documento assinado eletronicamente por **Anderson Ribeiro Duarte**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 04/09/2023, às 15:45, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0584873** e o código CRC **F4F721C0**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3559-1312 - www.ufop.br

Agradecimentos

Gostaria de expressar minha mais profunda gratidão às pessoas que tornaram este trabalho e minha jornada acadêmica possíveis:

- À minha mãe e meu pai, por acreditarem em mim incondicionalmente e fornecerem todo o suporte necessário para que eu pudesse concluir minha formação. Obrigado por serem minha base sólida.
- À minha irmã, cuja companhia e amizade têm sido fontes inestimáveis de conforto e inspiração.
- Ao meu orientador, pelo constante suporte, paciência e valiosos conselhos ao longo deste trabalho. Sua orientação foi fundamental para o sucesso deste projeto.
- Aos amigos que fiz durante o curso, que enriqueceram minha vida acadêmica e pessoal. Obrigado pela camaradagem e pelos bons momentos que tornaram essa jornada muito mais agradável.
- Por último, à República Refugiados, por ter sido minha segunda casa durante esses anos.

Resumo

Diversos processos presentes no cotidiano tem o comportamento análogo às redes de filas. Em diversas situações, o funcionamento destes processos envolve a alocação de somas financeiras vultosas. Diante disso, a alocação adequada de recursos, com interesse na obtenção de respostas eficiêntes e controle orçamentário são preponderantes. De fato, algumas dessas decisões, ainda no momento presente, são tomadas de forma meramente intuitiva. Diante disso, o fornecimento de mecanismos cientificamente subsidiados para fornecer estes resultados é de enorme valia. Particularmente na avaliação da alocação de recursos, o problema de alocação de servidores e dimensionamento das áreas de espera de filas dispostas em rede com interesse no funcionamento da rede com alta eficiência desperta grande interesse na comunidade científica e também para o mercado. Neste estudo, o clássico algoritmo heurístico de otimização *Simulated Annealing* foi utilizado como abordagem para alocação ótima de servidores e *buffers*. Uma implementação bastante detalhada e adaptável à novas abordagens foi apresentada.

Palavras-chave: Redes de filas, Servidores, *Buffers*, Otimização, *Simulated Annealing*.

Abstract

Several processes present in everyday life behave similarly to queueing networks. In several situations, the operation of these processes involves allocating large financial values. Given this, adequate resource allocation, with an interest in obtaining efficient responses and budgetary control, is preponderant. Some of these decisions, even at present, are made merely intuitively. Providing scientifically subsidized mechanisms to obtain these results is of enormous value. Particularly in evaluating resource allocation, the problem of server allocation and dimensioning of waiting areas of queues arranged in a network with an interest in the functioning of the network with high efficiency arouses great interest in the scientific community and the market. In this study, the classic heuristic optimization algorithm *Simulated Annealing* was used as an approach for optimal allocation of servers and *buffers*. A very detailed and adaptable implementation of new approaches was presented.

Keywords: Queueing networks, Servers, Buffers, Optimization, Simulated Annealing.

Lista de ilustrações

Sumário

1 Introdução

Os estudos que envolvem os problemas ligados à teoria das filas estão sempre associados com situações recorrentes da vida cotidiana de todos. Desde o fluxo de tráfego no trânsito até os sistemas de atendimento hospitalares, desde os mecanismos de atendimento virtual até os processos produtivos industriais, os sistemas de filas são presença marcantes na vida de todos os indivíduos. Para todo procedimento que segue um fluxo de tarefas subsequentes até um destino derradeiro, sempre existe uma modelagem adequada por meio de teoria de filas. Um desafio científico de grande complexidade reside na compreensa e melhora do funcionamento da dinâmica de funcionamento dessas filas.

Usualmente os sistemas de filas são representados por um modelo estocástico para a chegada de usuários que buscam por um determinado tipo de atendimento, e que posteriormente abandonam este sistema de filas após o atendimento de sua demanda. Além disso, a descrição de um modelo estocástico para delinear o tempo consumido para a prestação do referido serviço é de igual importância. Um exemplo bastante recorrente pode ser encontrada na área de computação, uma possível lista de tarefas e processos que aguarda para ser executada em uma unidade central de processamento, conforme discutido por Ahmed e Ouyang (2007) [1], Chen, Hu e Ji (2010) [7] e Inzillo, Rango e Quintana (2019) [16].

Os sistemas de filas podem prever atendimentos prestados por um servidor único ou dinâmicas multi-servidor. Em muitas situações, ocorrem filas com áreas de espera limitadas (*buffers* finitos) para um determinado serviço ofertado, têm-se o que usualmente estas são denominadas por filas finitas. Para filas finitas com espaço total para k clientes ($k-1$ em espera e um em atendimento), P_k denota a probabilidade de encontrar k usuários no sistema, com a inclusão daqueles que já estão em atendimento. Quando ocorre a chegada de um cliente em busca por serviço e as posições de espera estão todas ocupadas, o referido cliente é bloqueado pelo sistema. Diante disso, Cruz, Duarte; van Woesel (2008) [8] afirmam que altas probabilidades de bloqueio implicam na ineficiência do sistema de filas.

A Figura 1 apresenta uma esquematização específica para uma fila finita multiservidor. Neste contexto, λ_i representa a taxa de chegadas, existem $k-1$ locais de espera, e c servidores que atendem, todos, de acordo com uma taxa de serviço μ_i e a taxa de atendidos é dada por θ_i.

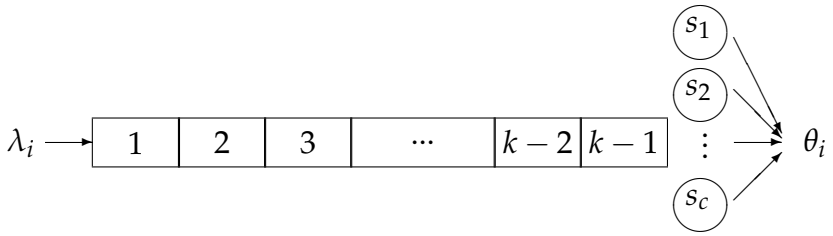

Figura 1 – Esquema ilustrativo de uma fila de servidores múltiplos com sua área de circulação.

1.1 Motivação

Um problema de grande aplicabilidade é a avaliação de métricas específicas capazes de mensurar a eficiência do funcionamento das redes de filas. A literatura apresenta diversas métricas adequadas para aferir níveis de eficácia da operação. Particularmente, a proposta de avaliar redes de filas multi-servidoras é de fato instigante. O próposito central está na tarefa de obter uma melhor estratégia para alocar áreas de espera e servidores em cada fila da rede, de forma que a performace do sistema seja otimizada.

Uma versão particular do problema de alocação de servidores em redes de filas foi apresentada por Duarte (2022) [14]. Um estudo que apresenta uma a abordagem que aplica o clássico algoritmo de otimização *Simulated Annealing* (SA). Estudos baseados no problema de alocação de áreas de espera também podem ser visualizados na literatura. Souza et al. (2020) [27] discutem este problema através de uma abordagem genética,por meio do clássico algoritmo NSGA-II (algoritmo genético de ordenação não-dominante elitista, do inglês *Elitist Non-dominated Sorting Genetic Algorithm*) de Deb et al. (2002) [12] O NSGA-II é composto por operadores de *seleção* e de *elitismo* que precisam estar especificamente estruturados para auxiliarem no processo de identificação de condições de otimalidade. Posteriormente, as soluções fornecidas pelo NSGA-II são pós-processadas por um algoritmo SA.

As duas investigações servem de motivação para a proposição de duas frentes de trabalho específicas:

i. formular um problema de otimização que contemple simultaneamente a otimização do consumo de recursos em áreas de espera e servidores alocados na rede de filas;

ii. implementar uma versão do algoritmo SA que se adapte bem à nova formulação do problema.

O problema de alocação de áreas de espera e servidores, trata de uma relação conflitante entre objetivos. A performance do sistema tende a melhorar com o acréscimo

de mais servidores e áreas de espera, o que é vantajoso. Porém, o acréscimo de servidores e áreas de espera acarreta em aumento de custos, o que não é desejável. Problemas de otimização com este tipo de conflito remetem a uma abordagem multiobjetivo.

1.2 Objetivos

As informações anteriores são abrangentes para garantir justificativa e motivação desse estudo. Os objetivos gerais e específicos são apresentados em sequência.

1.2.1 Objetivos Gerais

i. discutir os aspectos de utilização da heurística SA;

ii. obter uma implementação capaz de fornecer soluções factíveis quanto ao total de servidores alocados, áreas de espera alocadas, e com elevada taxa de atendimento dos clientes na rede de filas através das proposições presentes no estudo.

1.2.2 Objetivos Específicos

i. abordar formulações matemáticas para problemas de otimização.

ii. apresentar uma proposta de formulação do problema de otimização de alocação de servidores e áreas de espera;

ii. obter uma implementação capaz de fornecer soluções factíveis quanto ao total de servidores alocados, áreas de espera alocadas, e com elevada taxa de atendimento dos clientes na rede de filas através das proposições presentes no estudo.

2 Fundamentação Teórica

Os estudos acerca da teoria de filas tem sido objeto de investigação de diversos pesquisadores. O trabalho de Hillier (1963) [15] é de grande pioneirismo nessa área. Servidores separadas com áreas de espera finitas são abordadas. Os modelos de redes de filas descrevem várias situações de contexto da vida real, com grande interesse prático.

O foco desse trabalho está direcionado para a discussão do problema de alocação de servidores e áreas de circulação BSAP (do inglês, *buffer and server allocation problem*) que decorre dos problemas de alocação de áreas de circulação BAP (do inglês, *buffer allocation problem*) e de alocação de servidores SAP (do inglês *server allocation problem*). O BAP possui diversas possibilidades de formulação matemática, uma descrição bastante abrangente pode ser obtida no estudo de Cruz, Duarte e van Woensel (2008) [8]. Da mesma forma, o SAP também tem diversas estratégias de formulação, uma descrição sobre elas é apresentada por Duarte (2022) [14]. Por fim, o BSAP é discutido com detalhes por Martins et al. (2019) [19].

2.1 Teoria das Filas

As filas estão integradas ao cotidiano de todas as pessoas, sejamem grandes centros urbanos ou até mesmo em regiões menores e mais afastadas. As filas estão presentes nas relações do tráfego, nos atendimentos diversos (bancos, supermercados, alimentação, entre outros), na prestação de serviços médicos e muitos outros cenários. Além dessas relações que afetam diretamente aos indivídios, existem filas de impacto indireto, mas não menor, tais como: filas inerentes aos processos produtivos, sistemas logísticos, tarefas manufatureiras e muitas outras.

Em geral, os usuários dos sistemas tendem a acreditar, de forma até ingênua, que, somente eles se aborrecem com a existência das filas. Na prática, aqueles que oferecem serviços que geram filas também são prejudicados. A formação de filas, na maioria das situações, acarreta em alguma perda para os gestores desses serviços. Na maior parte das situações, a insatisfação decorrente dessas filas acarreta em perdas financeiras e de produtividade.

As filas surgem sempre no mesmo contexto, a capacidade de fornecimento de um determinado serviço não é capaz de suprir rapidamente a demanda existente pelo serviço. Logo, toda situação em que o recurso para oferecer tal serviço é limitado e a procura pelo referido serviço é estocástica, tem-se a formação de filas de espera. Isto ocorre por uma grande gama de razões: a dificuldade de implantar novos servidores;

a necessidade de maior tempo para execução do serviço; a espera da conclusão de outras etapas do processo. Em algumas situações, pode até mesmo ser economicamente inviável disponibilizar o serviço a todo tempo, isto de acordo com sua demanda. Pode também ocorrer alguma limitação do espaço físico disponibilizado para a instalação da estação de oferta do serviço em questão.

Na maior parte das situações, os problemas podem ser solucionados por meio de uma ampliação do investimento no sistema de filas. Por outro lado, a disponibilidade financeira para isso pode ser pequena ou até mesmo inexistente. Diante disso, o propósito central está em ser capaz de otimizar o sistema de filas em todas as suas nuances, ou seja, melhorar o desempenho de todas as tarefas inerentes ao seu funcionamento e minimizar a necessidade de recursos financeiros investidos para tal propósito. Isso leva a observar o problema por outro prisma: "qual o custo de espera do usuário?"; "quantos clientes podem esperar em cada fila?"; "qual o ganho em reduzir a oferta deixando clientes em fila?". Estudos associados à teoria das filas tentam, e em muitos casos com sucesso, através da utilização de recursos e ferramentas computacionais, matemáticas e estatísticas, responder a essas e várias outras perguntas.

A formação de sistemas de filas está diretamente ligada nas situações de incerteza no fluxo de produtos, usuários, tarefas entre outras possibilidades. As filas, do ponto de vista matemático, são determinadas com base em um mecanismo matemático capaz de descrever a estocasticidade inerente ao padrão de chegadas dos clientes, produtos, tarefas, etc. E também descrever a estocasticidade associada ao padrão do atendimento ao seviço requerido pelas chegadas.

2.1.1 Notação de Kendall

A notação descritiva para modelos de filas mais difundida é a notação proposta por Kendall [17]. A estrutura das filas é descrita por uma padrão representativo de letras. Este esquema fica mais claro ao observar as representações na Tabela 1.

Tabela 1 – Parametrização para filas de acordo com a notação de Kendall [17], adaptada de Souza (2020) [26].

parâmetro	símbolo	definição
A e B	M	Exponencial
	D	Determinístico
	E_p	Erlang$_p$ ($p = 1, 2, \dots$)
	H_p	Mistura de p Exponenciais
	G	Geral
m	$1, 2, \dots$	número de servidores
K	$1, 2, \dots$	espaço total (servidores + área de espera)

De forma resumida, as filas são descritas de forma resumida por $A/B/m/k$ em que: A descreve a distribuição de probabilidades para o tempo entre chegadas; B representa distribuição de probabilidades para o tempo de serviço; m ilustra o número de canais de serviço em paralelo; e k denota o espaço total disponibilizado para os clientes em espera mais o número de clientes que estão sendo atendidos.

Um exemplo simples seria considerar uma fila $M/D/2/\infty$ que indica um processo de fila com tempo entre chegadas com distribuição exponencial, tempo de serviço determinístico, dois servidores em paralelo para prestar os atendimento, e sem restrição no tamanho máximo da capacidade do sistema.

Quando é omitido o termo k, na notação de Kendall, entende-se que a fila tem capacidade infinita. Por exemplo, uma fila $M/G/1$ tem chegadas com distribuição exponencial, serviço com distribuição geral, ou seja, alguma distribuição de probabilidades genérica, um único servidor, e não há limite na capacidade do sistema e o atendimento é por ordem de chegada.

As características de um sistema de filas são basicamente:

i) Padrão de chegadas dos clientes;

ii) Padrão de serviço dos canais de atendimento;

iii) Capacidade do sistema;

iv) Número de canais de serviço;

v) Número de estágios de serviço.

Dessas características, fica claro que as duas primeiras estão associadas A demanda pelo serviço prestado e pela capacacidade de prestar o referido serviço. Por outro lado, particularmente o número de canais de serviço, ou seja, o número de servidores e a capacidade do sistema no que tange em capacidade de alocar indivíduos em áreas de espera são escolhas de gestão. Estas escolhas podem ser otimizadas para melhorar o funcionamento e também a reduzir o consumo de recurso.

A otimização de sistemas de rede de filas finitas é um ponto de profundo interesse. A capacidade de ajudar a compreender e aperfeiçoar vários sistemas presentes na vida das pessoas é motivadora e instigante. Potenciais usuários destes modelos de otimização baseados em redes de filas finitas incluem cientistas da computação, engenheiros de produção entre outros pesquisadores. Este modelos tendem a auxiliar na compreensão e melhoria de vários sistemas reais, incluindo sistemas de manufatura (Alves et al., 2011 [2]), de produção (Smith et al., 2010 [25]) e de saúde (Bruin et al., 2007 [3]), sistemas de tráfego de veículos e de pedestres (Cruz et al, 2005; Cruz et al.,

2008; Cruz et al., 2010 [8–10]), sistemas de computação e de comunicação (Ahmed e Ouyang, 2007; Chen, Hu e Ji, 2010; Inzillo et al., 2019 [1,7,16]), aplicações baseadas na *web*, configuradas em camadas (Chaudhuri et al., 2007 [6]) e com requisitos de qualidade de serviço (QoS) definidos em termos de tempo de resposta, taxa de atendimento (ou *throughput*), disponibilidade e segurança (Menasce, 2002 [20]), entre outros (Dimitirou e Lagaris, 2010; Osorio e Bierlaire, 2009 [13,21]).

2.2 Problemas de Otimização

O presente estudo tem interesse claro em otimização em sistemas de filas. Diante disso, algumas considerações são necessárias sobre problemas de otimização. Estudos de otimização podem incluir uma variedade grande de problemas cujo objetivo é buscar por alguma nível de otimalidade (Yang, 2010 [29]). Existem formas de classificação para os mais diversos problemas de otimização. Uma classificação resumida pode ser observada na Figura 2.

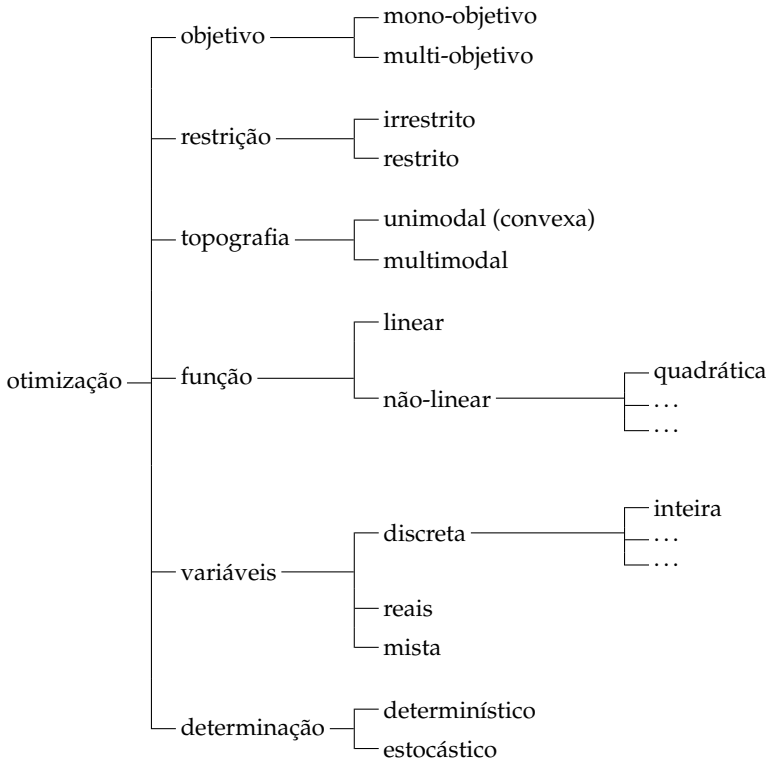

Figura 2 – Classificação dos problemas de otimização segundo Yang (2010) [29]

Este estudo aborda um problema *mono-objetivo*, *restrito*, *unimodal*, *não-linear geral*, *inteiro* e *estocástico*. Existem diversas técnicas para resolução de problemas de otimização. Essas técnicas variam significativamente de acordo com a natureza do problema. Ocorre portanto, uma exigência cada vez maior de conhecimento do problema e de um

vasto número de técnicas disponíveis. A complexidade de um problema de otimização depende, e muito, da sua função objetivo e do seu conjunto de restrições. Para essa linha de abordagem será descrito matematicamente um formato geral para os problemas de otimização:

$$\text{minimizar } f_i(\mathbf{x}), (i = 1, 2, \ldots, I),$$

sujeito a:

$$\begin{aligned} \phi_j(\mathbf{x}) &= 0, (j = 1, 2, \ldots, J), \\ \psi_k(\mathbf{x}) &\leqslant 0, (k = 1, 2, \ldots, K), \end{aligned}$$

em que $f_i(\mathbf{x}), \phi_j(\mathbf{x})$ e $\psi_k(\mathbf{x})$ são funções do vetor de decisão $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. Os componentes x_i de $\mathbf{x}$ são varáveis de decisão que podem ser contínuas, discretas ou uma mistura das duas. As funções $f_i(\mathbf{x})$, em que $i = 1, 2, \ldots, I$, são denominadas funções objetivo. O espaço das variáveis de decisão é denominado espaço de busca, enquanto o espaço formado pelos valores das funções objetivo é denominado espaço das soluções.

As funções objetivo podem ser lineares ou não lineares. As igualdades para ϕ_j e desigualdades para ψ_k são denominadas restrições. Os problemas de otimização podem ter naturezas mono ou multi-objetivo.

2.3 O Problema de Alocação de Áreas de Circulação e Servidores

Especificamente neste estudo, o alvo é direcionado para um problema de buscar uma alocação eficiente para as áeras de espera e também os servidores (do inglês, *Buffer and Server allocation problem*, o BSAP). As redes de filas objeto desse estudo, usualmente são bem representadas através de um grafo $\mathcal{G}(V, A)$ em que V é um conjunto finito de vértices (filas) e A é um conjunto finito de arestas (conexões entre as filas). A rede complexa de filas, apresentada na Figura 3, foi adaptada da literatura (Smith e Cruz, 2005 [24]).

Esta rede é bastante adequada para diversos experimentos de estudo. Isso porque inclui as situações topológicas de fusão, divisão e série. Para as discussões de otimização da alocação de recursos em redes de filas, a formulação matemática para o problema em investigação é uma etapa bastante relevante. O interesse aqui é discutir as possibilidades de formulação para o problema de otimização da utilização de recursos em redes acíclicas de filas. Particularmente foi utilizada uma formulação mono-objetivo para o BSAP. É importante salientar que as formulações não são únicas. Em suma são dependentes do interesse do pesquisador e das necessidades de projeto. Entretanto, a

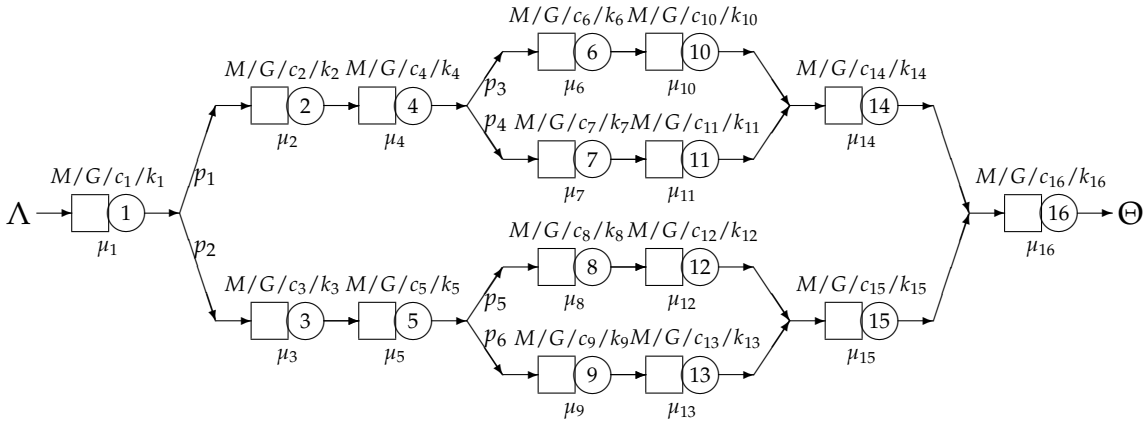

Figura 3 – Rede de filas com topologia mista adaptada de Smith e Cruz (2005) [24].

literatura apresenta algumas formulações mais recorrentes e os problemas possuem algumas formulações mais gerais.

2.3.1 Formulação para o BSAP

A literatura apresenta diversas formulações de estudo para o BSAP, sejam elas multiobjetivo, ou mono-objetivo. Aqui o BSAP é apresentado por meio de uma formulação mono-objetivo:

$$\text{minimizar} \sum_{i=1}^{m} \alpha_i b_i + (1 - \alpha_i) c_i, \tag{2.1}$$

sujeito a:

$$\begin{aligned} &\Theta(\mathbf{B}, \mathbf{C}) \geqslant \Theta_{\min}, \\ &0 \leqslant \alpha_i \leqslant 1, \forall i \in \{1, 2, \ldots, m\}, \\ &b_i \in \mathbb{N}, \forall i \in \{1, 2, \ldots, m\}, \\ &c_i \in \mathbb{N}^*, \forall i \in \{1, 2, \ldots, m\}, \end{aligned} \tag{2.2}$$

em que b_i representa a quantidade de *buffers* alocados para a i-ésima fila da rede de filas, c_i representa a quantidade de servidores alocados para a i-ésima fila da rede de filas, $\mathbf{B}$ é o vetor de alocações de *buffers*, $\mathbf{C}$ é o vetor de alocações de servidores, as constantes α_i e $1 - \alpha_i$ definem a relação de proporcionalidade entre o custo de *buffers* e servidores e $\Theta(\mathbf{B}, \mathbf{C})$ é a taxa de atendimento da rede de filas. O objetivo a função associada ao consumo de recursos em *buffers* e servidores em uma rede com m filas, sujeito a um limiar de atendimento $\Theta_{\min}$.

A taxa de atendimentos $\Theta(\mathbf{B}, \mathbf{C})$ de uma rede de filas é dependente da taxa de atendimento inerente a cada uma das filas da rede. Para uma fila individual com k espaços de circulação, c servidores e $b = k - c$ áreas de espera, a taxa de atendimento é a proporção da taxa de entrada λ de clientes que de fato serão atendidos. É importante

observar que dada a restrição de área de espera, existe uma probabilidade de um cliente qualquer encontra a fila bloqueada, ou seja, todos os lugares de atendimento e espera estão em uso, nessa situação, este cliente não poderá ser atendido.

A probabilidade de um cliente encontra a fila bloqueada é usualmente denominada probabilidade de bloqueio p_k o termo p_k remete a probabilidade k usuários já estarem na fila (aqueles em serviço somados com aqueles em espera). Este estudo tem o seu foco centrado em filas de chegada e atendimento markovianos, ou seja, filas $M/M/c/k$. Para tanto a probabilidade de bloqueio pode ser obtida por:

$$p_k = \left(\frac{r^c \rho^{k-c}}{c!} \right) p_0,$$

em que $r = {}^{\lambda}/_{\mu}$, $\rho = {}^{\lambda}/_{c\mu}$, λ é a taxa de chegada na fila, μ é a taxa de serviço dos c servidores e p_0 é a probabilidade de nenhum usuário na fila e nos sevidores. A probabilidade p_0 é dada por:

$$p_0 = \begin{cases} \dfrac{1}{\left(1 + \sum_{j=1}^{c} \dfrac{r^j}{j!} + \dfrac{r^c \rho (1 - \rho^{k-c})}{c!(1-\rho)} \right)} & \text{se } \rho \neq 1 \\ \dfrac{1}{\left(1 + \sum_{j=1}^{c} \dfrac{r^j}{j!} + \dfrac{r^c (k - c + 1)}{c!} \right)} & \text{se } \rho = 1. \end{cases}$$

Diante disso, para uma fila individual (a i-ésima fila de uma rede de filas), a taxa de atendimentos θ_i é dada por $\theta_i = (1 - p_{k_i})\lambda_i$. Dada uma rede de filas qualquer, a taxa de chegadas na rede Λ é na verdade a taxa de chegada na(s) fila(s) inicial(is) da rede de filas. Se é uma única fila inicial, então $\lambda_1 = \Lambda$, se existem mais filas iniciais, a taxa Λ é dividida entre as diversas filas de entrada de acordo com alguma regra de proporcionalidade fixada. Para as demais filas da rede, a taxa de chegadas é desconhecida, mas pode ser obtida pela relação $\theta_i = (1 - p_{k_i})\lambda_i$ aplicada nas filas predecessoras.

Ao aplicar este raciocínio para todas as filas da rede é possível obter a taxa de atendimentos $\Theta(\mathbf{B}, \mathbf{C})$ de uma rede de filas. Essa taxa é dada pela taxa de atendimentos da última fila do fluxo da rede. Na eventualidade de existirem mais de uma fila de saída da rede, a soma das taxas de atendimento dessas filas determina a taxa de atendimentos geral da rede de filas.

2.3.2 Algoritmo de otimização *Simulated Annealing*

A escolha por uma heurística de otimização decorre de existência de um elevado número de configurações possíveis para serem avaliadas para redes de filas. Uma busca

exaustiva se tornaria inviável do ponto de vista computacional. Uma heurística de otimização capaz de se adaptar bem ao problema em investigação é clássico algoritmo *Simulated Annealing*.

O algoritmo *Simulated Annealing* foi descrito inicialmente por Kirkpatrick et al. 1983 [18] e Černỳ (1985) [5]. É um algoritmo inspirado no processo de arrefecimento de sistemas físicos. Os princípios básicos têm origens em termodinâmica estatística, uma analogia com o recozimento (arrefecimento controlado) de sólidos poderia fornecer uma estrutura para o desenvolvimento de um algoritmo genérico de otimização capaz de escapar de ótimos locais na busca pelo ótimo global. Desde a sua introdução, como um método de otimização combinatorial.

De uma forma simplista, o método depende de um funcional objetivo de otimização e de um critério de vizinhança entre as soluções candidatas. Busca-se reproduzir uma cadeia markoviana cujo espaço de estados e composto por um conjunto de possíveis soluções para o problema de otimização em estudo.

O *Simulated Annealing* opera da seguinte forma: se o $n-$ésimo estado da cadeia de Markov é uma possível solução S_1, então alguma solução vizinha da solução S_1 é selecionada aleatoriamente; se o estado vizinho escolhido for S_2, então o próximo estado da cadeia será S_2, se este for superior a S_1 avaliado através do funcional objetivo do problema. Caso contrário, o próximo estado da cadeia ainda poderá ser S_2 com uma probabilidade p, ou então a cadeia se manterá em S_1 com probabilidade $1 - p$. A escolha do valor p, em geral, é dependente do número de passos já executados pela cadeia de Markov e também pelo acréscimo ou decréscimo na função objetivo, gerado pela possível troca entre as soluções S_1 e S_2. Uma escolha computacionalmente usual de p é $e^{C \log(1+n)}$ em que a constante C é dada por $C = -|f(S_1) - f(S_2)|$ (com f a função objetivo definida por 2.1) e n é o número de passos dados pela cadeia de Markov até o instante corrente.

Ao gerar um conjunto de m passos sucessivos da cadeia, $\mathcal{S} = \{S_1, \ldots, S_m\}$, pode-se estimar a solução ótima $(b_1, b_2, \ldots, b_m, c_1, c_2, \ldots, c_m)$ que minimiza a função objetivo (2.1) com respeito ao conjunto $\mathcal{S}$. A proposta em estudo é de uma busca dentre as possíveis alocações no vetor $(b_1, b_2, \ldots, b_m, c_1, c_2, \ldots, c_m)$ por meio do algoritmo *Simulated Annealing*, com interesse em obter a configuração que minimiza a função objetivo (2.1) em termos de uma taxa de atendimento mínima préfixada Θ_{min}. É importante a definição do conceito de vizinhança entre as possíveis alocações para o vetor $(b_1, b_2, \ldots, b_m, c_1, c_2, \ldots, c_m)$.

Alguns critérios de vizinhança foram estabelecidos, seja $x = (b_1, \ldots, b_i, \ldots, b_j, \ldots, b_m, c_1, c_2, \ldots, c_m)$, considere a escolha aleatória de inteiros $i, j \in \{1, \ldots, m\}$ então a solução $x^* = (b_1, \ldots, b_j, \ldots, b_i, \ldots, b_m, c_1, c_2, \ldots, c_m)$ é uma solução vizinha de x. Note que x^* difere de x por trocar a alocação de *buffers* entre as filas i e j da rede de filas.

Um segundo critério utilizado foi, seja $x = (b_1, \ldots, b_i, \ldots, b_m, c_1, c_2, \ldots, c_m)$, considere a escolha aleatória do inteiro $i \in \{1, \ldots, m\}$ então $x^* = (b_1, \ldots, b_i + 1, \ldots, b_m, c_1, c_2, \ldots, c_m)$ é uma solução vizinha de x. Note que x^* difere de x por acrescentar uma unidade de *buffer* na fila i da rede de filas.

Um terceiro critério utilizado foi, seja a solução dada por $x = (b_1, \ldots, b_i, \ldots, b_m, c_1, c_2, \ldots, c_m)$, considere a escolha aleatória do inteiro $i \in \{1, \ldots, m\}$ então a solução $x^* = (b_1, \ldots, b_i - 1, \ldots, b_m, c_1, c_2, \ldots, c_m)$ é uma solução vizinha de x. Note que x^* difere de x por retirar uma unidade de *buffer* na fila i da rede de filas. É importante observar que este terceiro critério somente pode ser utilizado se preserva $b_i \geqslant 0$ que é uma condição de factibilidade da solução vizinha.

Analogamente critérios similares foram estabelecidos para alteração na alocação de servidores. Seja $x = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_i, \ldots, c_j, \ldots, c_m)$, considere a escolha aleatória de inteiros $i, j \in \{1, \ldots, m\}$ então $x^* = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_j, \ldots, c_i, \ldots, c_m)$ é uma solução vizinha de x. Note que x^* difere de x por trocar a alocação de servidores entre as filas i e j da rede de filas.

Um segundo critério utilizado foi, seja a solução dada por $x = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_i, \ldots, c_m)$ considere a escolha aleatória do valor inteiro i tal que $i \in \{1, \ldots, m\}$ então $x^* = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_i + 1, \ldots, c_m)$ é uma solução vizinha de x. Note que x^* difere de x por acrescentar uma unidade de servidor na fila i da rede de filas.

Um terceiro critério utilizado foi, seja a solução $x = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_i, \ldots, c_m)$ considere a escolha aleatória do inteiro $i \in \{1, \ldots, m\}$ então $x^* = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_i - 1, \ldots, c_m)$ é uma solução vizinha de x. Note que x^* difere de x por retirar uma unidade de servidor na fila i da rede de filas. É importante observar que este terceiro critério somente pode ser utilizado se preserva $c_i \geqslant 1$ que é uma condição de factibilidade da solução vizinha.

O algoritmo depende de critérios probabilísitcos para a escolha de utilização de cada um dos possíveis critérios de vizinhança. A implementação que será apresentada deixa a probabilidade para utilização de cada uma das estruturas de vizinhança como um parâmetro de entrada para o algoritmo. O algoritmo *Simulated Annealing* implementado é mostrado no Algoritmo 1.

Algoritmo 1: Pseudo-código para o algoritmo *Simulated Annealing*.

```
/* gere um conjunto de soluções iniciais {S_1, S_2, ..., S_w} */
for j = 1; j ⩽ w, j++ do
    i ← 1
    while i > i_max do
        if i = 1 then
            sol_max ← sol_i ← C_j
        end
        U ← uniform(0, 1)
        /* utiliza U para escolher o critério de vizinhança */
        sol_aux ← PerturbSolution(sol_i)
        if sol_aux é superior a sol_i then
            sol_{i+1} ← sol_aux
            if sol_aux é superior a sol_max then
                sol_max ← sol_aux
            end
        else
            U ← uniform(0, 1)
            Δ ← |F(sol_i) − F(sol_aux)|
            if U < e^{−Δ log(1+i)} then
            end
            sol_{i+1} ← sol_aux
        end
        i ← i + 1
    end
    sol_final_j ← sol_max
end
return sol_final
```

3 Resultados Alcançados

O problema de investigação em filas $M/M/c/k$ é modelado por meio de um grafo direcionado e valorado. Nessa abordagem, cada vértice do grafo representa uma fila com características bem definidas, que incluem uma taxa de serviço λ, um número de servidores c, e um tamanho total da área alocada para a fila k, constituído pela soma do número de *buffers* e do número de servidores. Adicionalmente, cada vértice possui atributos que indicam se ele atua como um nó de origem ou um nó de destino, isso permite que o modelo acomode cenários em que o fluxo de trabalho começa e termina em pontos específicos.

As arestas do grafo estabelecem a conectividade entre as filas. O peso associado a cada aresta representa a probabilidade de uma tarefao (ou cliente) ser transferido de uma fila para outra, isso permite que o modelo acomode sistemas em que o fluxo de trabalho não se distribui de forma uniforme entre as filas.

Um aspecto notável deste modelo é sua adaptabilidade. Os parâmetros para as filas e suas conexões são extraídos de um arquivo de entrada, que contém informações suficientes para modelar uma ampla gama de cenários. Essa característica torna o modelo altamente flexível, pois elimina a necessidade de modificar o código-fonte para estudar diferentes configurações de sistema.

No contexto deste modelo, métricas de desempenho específicas são calculadas para cada fila. Essas métricas compreendem a taxa de chegada λ, o fluxo de trabalho efetivo θ, a probabilidade de bloqueio p_k e o tempo médio de espera na fila W. Elas fornecem informações valiosas para a avaliação do desempenho do sistema e são cruciais para a subsequente fase de otimização. O algoritmo de otimização *Simulated Annealing* é utilizado para encontrar uma alocação eficiente de servidores e buffers, com interesse em minimizar um custo total definido, enquanto mantém θ acima de um limiar mínimo.

Com interesse em ilustrar e elucidar toda a tarefa de implementação desenvolvida neste estudo, uma documentação detalhada será apresentada.

3.1 Documentação do Código

A documentação apresentada neste estudo separa as informações em dois setores principais, os pacotes já desenvolvidos que são utilizados e as funções implementadas exclusivamente para este estudo.

3.1.1 Dependências de Pacotes

3.1.1.1 Pacote `stringr`

Descrição: O pacote `stringr` [28] é uma coleção de funções que tornam a manipulação de strings mais fácil e consistente em R.

Utilização no Código: Foi usado principalmente para manipulação e tratamento de strings, especialmente durante a leitura e processamento dos arquivos de entrada que contêm as configurações do sistema de filas.

3.1.1.2 Pacote `ggraph`

Descrição: O pacote `ggraph` [22] é uma extensão do pacote `ggplot2`, dedicado à visualização de grafos e redes.

Utilização no Código: Utilizado para visualizar a topologia do grafo que representa o sistema de filas, o que facilita a compreensão e o *debug* do modelo.

3.1.1.3 Pacote `tidygraph`

Descrição: O pacote `tidygraph` [23] oferece uma abordagem "tidy" para a manipulação de objetos de grafo.

Utilização no Código: Utilizado para manipular e consultar objetos de grafo de forma eficiente, como a extração de vizinhos ou subgrafos.

3.1.1.4 Pacote `igraph`

Descrição: O pacote `igraph` [**?**, 11] é uma coleção de ferramentas de manipulação e análise de redes para R.

Utilização no Código: Foi usado extensivamente para criar e manipular o grafo que representa o sistema de filas. A criação de vértices, arestas e atributos associados são feitos usando este pacote.

3.1.1.5 Pacote `queueing`

Descrição: O pacote `queueing` [4] contém rotinas para calcular métricas para diferentes tipos de filas.

Utilização no Código: Utilizado para calcular todos os possíveis caminhos do circuito de filas.

3.1.2 Funções e Procedimentos Implementados

3.1.2.1 Função `calc_PK`

Descrição: A função `calc_PK` é responsável por calcular a probabilidade de bloqueio P_k em um sistema de filas M/M/c/K.

Parâmetros:

- `lambda`: taxa de chegada (λ).
- `mu`: taxa de serviço (μ).
- `c`: número de servidores (c).
- `k`: tamanho total da fila (k).

Variáveis internas:

- `r`: intensidade de tráfego no servidor, calculada como $r = \lambda/\mu$.
- `rho`: intensidade de tráfego, calculada como $\rho = r/c$.
- `sum_j`: soma auxiliar usada no cálculo do denominador na fórmula de P_k.

Algoritmo:

1. Calcula-se `r` e `rho` usando as taxas de chegada e de serviço, bem como o número de servidores.
2. O algoritmo entra em um condicional para calcular `sum_j` com base no número de servidores c.
3. Utiliza-se um segundo condicional para calcular P_k de acordo com o valor de ρ.

Valor de retorno: A função retorna o valor calculado de P_k, que é a probabilidade de bloqueio no sistema.

Exemplo de uso: Suponha um sistema de filas com os seguintes parâmetros:

- $\lambda = 10$ (clientes por minuto)
- $\mu = 2$ (clientes atendidos por minuto por servidor)
- $c = 3$ servidores
- $k = 20$ tamanho total da fila

A diretiva `calc_PK(10, 2, 3, 20)` retornará o valor de P_k para uma fila com estes parâmetros.

3.1.2.2 Função `calc_P0`

Descrição: A função `calc_P0` é responsável por calcular a probabilidade P_0 de um sistema de filas estar vazio, ou seja, sem clientes na fila e servidores ociosos.

Parâmetros:

- `lambda`: taxa de chegada (λ).
- `mu`: taxa de serviço (μ).
- `c`: número de servidores (c).
- `k`: tamanho total da fila (k).

Variáveis internas:

- `r`: intensidade de tráfego no servidor, calculada como $r = \lambda/\mu$.
- `rho`: intensidade de tráfego, calculada como $\rho = r/c$.
- `sum_j`: soma auxiliar usada no cálculo do denominador na fórmula de P_0.

Algoritmo:

1. Calculam-se `r` e `rho` usando as taxas de chegada e de serviço, bem como o número de servidores.
2. Utiliza-se um condicional para calcular P_0 com base no valor de ρ.

Valor de retorno: A função retorna o valor calculado de P_0, que é a probabilidade de não existerem usuários em nenhum servidor e nenhum local de espera na fila, ou seja, *buffer* vazio.

Exemplo de uso: Suponha um sistema de filas com os seguintes parâmetros:

- $\lambda = 10$ (clientes por minuto)
- $\mu = 2$ (clientes atendidos por minuto por servidor)
- $c = 3$ servidores
- $k = 20$ tamanho total da fila

A diretiva `calc_P0(10, 2, 3, 20)` retornará o valor de P_0 para uma fila com estes parâmetros.

3.1.2.3 Função `calc_W`

Descrição: A função `calc_W` é responsável por calcular o tempo médio de espera W na fila em um sistema com filas $M/M/c/k$.

Parâmetros:

- `lambda`: taxa de chegada (λ).
- `mu`: taxa de serviço (μ).
- `c`: número de servidores (c).
- `k`: tamanho total da fila (k).
- `PK`: Probabilidade de bloqueio (P_k).
- `P0`: Probabilidade de o sistema estar vazio (P_0).

Variáveis internas:

- `r`: intensidade de tráfego no servidor, calculada como $r = \lambda/\mu$.
- `rho`: intensidade de tráfego, calculada como $\rho = r/c$.
- `num_1`, `num_2`: Variáveis auxiliares para o numerador da fórmula de W.
- `den_1`, `den_2`: Variáveis auxiliares para o denominador da fórmula de W.

Algoritmo:

1. Calculam-se `r` e `rho` usando as taxas de chegada e de serviço, bem como o número de servidores.
2. Utiliza-se um condicional para calcular W com base em `rho`, `PK`, e `P0`.

Valor de retorno: A função retorna o valor calculado de W, que é o tempo médio de espera na fila.

Exemplo de uso: Suponha um sistema de filas com os seguintes parâmetros:

- $\lambda = 10$ (clientes por minuto)
- $\mu = 2$ (clientes atendidos por minuto por servidor)
- $c = 3$ servidores
- $K = 20$ tamanho total da fila

- $P_k = 0.05$ probabilidade de bloqueio
- $P0 = 0.1$ probabilidade de o sistema estar vazio

Chamar `calc_W(10, 2, 3, 20, 0.05, 0.1)` retornará o valor de W para uma fila com estes parâmetros.

3.1.2.4 Procedimento `ETL`

Descrição: O procedimento `ETL` é responsável pela extração, transformação e carregamento (ETL) de dados do sistema de filas a partir de um arquivo de texto. Os dados extraídos incluem informações sobre as filas, os nós de origem e destino, e as taxas de serviço. É gerado uma variável *grafo* que representa o grafo associado à rede de filas.

Parâmetros:

- `caminho`: Caminho do arquivo de texto que contém os parâmetros do sistema de filas.

Variáveis internas:

- `linhas, data`: Armazenam as linhas do arquivo após a leitura e limpeza.
- `parametros`: Lista que contém os parâmetros extraídos do arquivo.
- `NQ, NARCS, NSNODE, NEND, ENDD`: Parâmetros globais extraídos.
- `matriz_nodes, matriz_arrival, matriz_service`: Matrizes que armazenam informações sobre as filas, nós de origem e taxas de serviço.
- `grafo`: Objeto do grafo que representa o circuito de filas.

Algoritmo:

1. Lê o arquivo de texto e armazena as linhas em `linhas`.
2. Realiza limpeza e transformação dos dados.
3. Extrai e armazena os parâmetros globais e as matrizes de informação.
4. Inicializa o grafo e seus atributos.
5. Identifica e armazena todos os caminhos possíveis no grafo.

Valor de retorno: Por se tratar de um procedimento, ETL não possui um valor de retorno explícito, mas configura várias variáveis globais que são usadas posteriormente para modelar o sistema de filas.

3.1.2.5 Função `calc_Theta`

Descrição: A função `calc_Theta` é responsável por calcular o fluxo de trabalho efetivo (θ) para cada fila no sistema de filas modelado como um grafo. O cálculo é feito com base nas taxas de chegada e serviço, bem como nas probabilidades de bloqueio.

Parâmetros:

- `grafo`: Objeto de grafo que representa o sistema de filas.
- `matriz_arrival`: Matriz que contém informações sobre os nós de origem e suas respectivas taxas de chegada.

Variáveis internas:

- `mu, c, K, lambda`: taxa de serviço, número de servidores, tamanho total da fila e taxa de chegada, respectivamente.
- `edges_in, neighbors_in`: Arestas e vizinhos que chegam ao nó atual.
- `theta_final`: fluxo de trabalho efetivo acumulado dos nós finais.

Algoritmo:

1. Percorre cada vértice do grafo.
2. Calcula e atualiza os atributos `lambda`, `PK`, `P0`, `W` e `theta` para cada vértice.
3. Acumula o valor de θ para os nós finais.

Valor de retorno: A função retorna o fluxo de trabalho efetivo acumulado (θ) dos nós finais.

Exemplo de uso: Suponha um grafo nominado "grafo" e uma matriz denominada "matriz_arrival" que contém as taxas de chegada para os nós de origem. A diretiva `calc_Theta(grafo, matriz_arrival)` retornará o fluxo de trabalho efetivo acumulado (θ) para a saída da rede de filas.

3.1.2.6 Função `calc_TotalW`

Descrição: A função `calc_TotalW` é responsável por calcular o tempo médio de espera total (W_{total}) na rede de filas modelada como um grafo. Esta métrica é calculada com base nos tempos médios de espera (W) de cada fila e nas probabilidades de roteamento entre elas.

Parâmetros:

- `grafo`: objeto de grafo que representa o sistema de filas.
- `caminhos`: matriz que contém todos os possíveis caminhos entre os nós de origem e os nós de destino.

Variáveis internas:

- `Wtotal`: tempo médio de espera total inicializado como zero.
- `peso_caminho`: peso associado a cada caminho, calculado com base nas probabilidades de roteamento das arestas.
- `edge`: identificador da aresta que liga dois nós consecutivos em um caminho.

Algoritmo:

1. Inicializa W_{total} como zero.
2. Percorre todos os caminhos possíveis.
3. Para cada caminho, calcula seu peso multiplicando as probabilidades de roteamento das arestas.
4. Soma ao W_{total} o produto do peso do caminho pelo somatório dos W dos nós no caminho.

Valor de retorno: A função retorna W_{total}, o tempo médio de espera total do sistema de filas.

Exemplo de uso: Suponha um grafo nominado "grafo" e uma matriz denominada "matriz_arrival" que contém as taxas de chegada para os nós de origem. A diretiva `calc_TotalW(grafo, caminhos)` retornará W_{total}.

3.1.2.7 Função `initial_Solutions`

Descrição: A função `initial_Solutions` gera soluções iniciais pseudo-aleatórias para a configuração de servidores e *buffers* do sistema de filas. Ela busca garantir que a métrica θ (fluxo de trabalho efetivo) permaneça acima de um limite especificado.

Parâmetros:

- `grafo`: Objeto de grafo que representa o sistema de filas.

- `matriz_arrival`: Matriz contendo informações sobre os nós de origem e as taxas de chegada.
- `n_starts`: número de soluções iniciais a serem geradas.
- `theta_restrict`: valor mínimo para o fluxo de trabalho efetivo θ.
- `correct_buffers`: fator de correção para a quantidade total de buffers.
- `correct_servers`: fator de correção para a quantidade total de servidores.

Variáveis internas:

- `Tot_buffers`: quantidade total de buffers.
- `Tot_servers`: quantidade total de servidores.
- `buffers`: vetor com a distribuição de *buffers* para cada fila.
- `servers`: vetor com a distribuição de servidores para cada fila.
- `theta`: fluxo de trabalho efetivo.

Algoritmo:

1. Determina a quantidade total de *buffers* e servidores com base no grafo e nos fatores de correção.
2. Gera soluções iniciais pseudoaleatórias considerando restrições em θ.
3. Para cada solução inicial, ajusta os *buffers* e servidores e recalcula θ até que satisfaça a restrição.
4. Agrupa todas as soluções iniciais em uma matriz.

Valor de retorno: A função retorna uma matriz com as soluções iniciais para *buffers* e servidores de todas as filas, onde os NQ valores iniciais de uma coluna representam os *buffers* para a fila i e os NQ valores finais representam os servidores.

Exemplo de uso: Suponha um grafo nominado "grafo" e uma matriz denominada "matriz_arrival" que contém as taxas de chegada para os nós de origem. Suponha ainda, gerar 10 soluções iniciais que garantam θ acima de 90% do theta inicial, com fatores de correção 10 para *buffers* e 10 para servidores. A diretiva `initial_Solutions(grafo, matriz_arrival, 10, 0.9, 10, 10)` retornará sugestões de soluções iniciais que respeitem o parâmetro `theta_restrict`.

3.1.2.8 Função `calc_Cost`

Descrição: A função `calc_Cost` calcula o custo total do sistema de filas com base em uma solução fornecida para a configuração de servidores e buffers. O custo é uma função linear dos números de servidores e buffers, ponderada por um fator de relação de custo entre áreas de espera e servidores α.

Parâmetros:

- `alpha`: fator de ponderação que equilibra o custo dos servidores e dos buffers.
- `solution`: vetor que contém a configuração atual de servidores e *buffers* para todas as filas.
- `NQ`: número de filas no sistema.

Variáveis internas:

- `cost`: Custo total calculado.

Algoritmo:

1. Calcula o custo total como uma combinação linear dos números de servidores e buffers, ponderada por α.

Valor de retorno: A função retorna o custo total calculado para a configuração fornecida.

Exemplo de uso: Suponha uma solução denominada "solution" que especifica a configuração de servidores e *buffers* para um sistema com 3 filas. Se o objetivo for equilibrar o custo dos servidores e *buffers* com $\alpha = 0{,}5$, então a diretiva `calc_Cost(0.5, solution, 3)` retornará o custo total para essa configuração.

3.1.2.9 Função `SA` (Simulated Annealing)

Descrição: A função `SA` implementa o algoritmo *Simulated Annealing* para otimizar a configuração de servidores e *buffers* em um sistema de filas em rede. Ela visa minimizar o custo total do sistema enquanto mantém um determinado nível de desempenho, dado pelo valor de θ.

Parâmetros:

- `grafo1`: o grafo que representa o sistema de filas.
- `sol_ini`: solução inicial para a configuração de servidores e buffers.
- `NQ`: número de filas no sistema.

- `theta_restrict`: valor mínimo de θ que deve ser mantido.
- `cont_max`: número máximo de iterações do algoritmo.

Variáveis internas:

- `alpha`: fator de ponderação para o cálculo do custo.
- `p_viz_1, p_viz_2, ...`: probabilidades para escolher diferentes critérios de vizinhança.
- `F_max, F_new, F_cur`: valores de custo para as soluções máxima, nova e atual.
- `sol_max, sol_new, sol_cur`: soluções para máxima, nova e atual.
- `theta_new`: novo valor calculado de θ.
- `W_new, W_cur`: novo valor do tempo médio de espera na fila.
- `W_new, W_cur`: valor corrente do tempo médio de espera na fila.

Algoritmo:

1. Inicializa as variáveis e parâmetros.
2. Enquanto a condição de parada não for atingida, faça:
   a) Escolha um critério de vizinhança com base em uma variável aleatória u_{viz}.
      - Se u_{viz} está dentro de um intervalo específico, um critério de vizinhança é escolhido.
   b) Gere uma nova solução com base no critério de vizinhança escolhido.
      - Troca entre *buffers*
      - Adição de um *buffer*
      - Remoção de um *buffer*
      - Troca entre servidores
      - Adição de um servidor
      - Remoção de um servidor
   c) Avalie a nova solução e atualize a solução atual e a solução máxima conforme necessário:
      - Calcule o custo da nova solução (F_{new}) e da solução atual (F_{cur}).
      - Se $F_{\text{new}} \leqslant F_{\text{cur}}$, atualize a solução atual e, se necessário, a solução máxima.

- Se $F_{\text{new}} > F_{\text{cur}}$, gere um número aleatório u entre 0 e 1. Se u é menor ou igual a $\text{cont}^{(F_{\text{cur}} - F_{\text{new}})}$, aceite a nova solução. Esta condição permite que o algoritmo aceite soluções piores com uma probabilidade decrescente, e evita que ele fique preso em mínimos locais, o que permite uma exploração mais ampla do espaço de soluções, principalmente nas iterações iniciais. O termo "cont" é um contador que aumenta a cada iteração, isso representa uma forma de "resfriamento" do sistema, e reduz a probabilidade de aceitar soluções piores à medida que o algoritmo avança.

Valor de retorno: A função retorna a melhor solução encontrada para a configuração de servidores e *buffers* fornecida, ou seja, para cada solução inicial dada.

Exemplo de uso: Suponha um grafo nominado "grafo" que represente a rede de filas, , uma solução inicial 'sol_ini'", e que o objetivo seja um número máximo de 1000 iterações do algoritmo. Para tanto, a diretiva `SA(grafo, sol_ini, 3, 0.9, 1000)`, em que 3 é o número de filas da rede em estudo, e 0.9 é a porcentagem mínima do valor de theta inicial aceitável para o novo θ, ou seja, a restrição ao valor de θ para a nova solução fornecida pelo algoritmo.

A metodologia decorrente da implementação descrita neste texto tem como objetivo otimizar uma série de soluções iniciais utilizando a heurística *Simulated Annealing*.

Inicialmente, uma entrada de dados de descrição da rede de filas é carregada através da função `ETL`, o usuário tem liberdade para estabelecer diversas redes de filas desde que seguido a estrutura de dados descrita no atual formato dos arquivos de entrada. A função `ETL` é responsável por extrair, transformar e carregar os dados desse arquivo para uso posterior no algoritmo.

Na etapa de geração das soluções iniciais, algumas variáveis e parâmetros são inicializados. A função `initial_Solutions` é utilizada para criar um conjunto de soluções iniciais com base nos parâmetros fornecidos, como quantidade de soluções, restrição para o valor de θ e correções para *buffers* e servidores.

De posse de um conjunto de soluções iniciais, o algoritmo entra em sua fase de otimização. Para cada solução inicial gerada, a função `SA` (*Simulated Annealing*) é invocada para refinar e melhorar essa solução. As soluções otimizadas são armazenadas em uma matriz *output* denominada `sol_fim`.

Por fim, foi implementado um mecanismo de verificação de qualidade para cada solução, uma comparação entre a solução inicial e a solução final otimizada fornecida. Para tanto, o custo associado a cada solução é calculado por meio da função `calc_Cost` e os resultados são impressos. Além disso, o valor de θ para cada solução é calculado com a função `calc_Theta`, e os resultados também são impressos. Esta etapa permite uma avaliação quantitativa das melhorias obtidas através da heurística *Simulated Annealing*.

O Anexo A apresenta uma versão resumida da implementação, sem as funções de cálculo de parâmetros da rede de filas. Estes códigos podem ser fornecidos pelos autores sob solicitação, desde que sejam utilizados para fins unicamente acadêmicos.

4 Considerações Finais

O estudo e a implementação de uma aplicação computacional para otimização em redes de filas ofereceram novas perspectivas sobre a eficiência e otimização de sistemas em fila. Além disso, apresentaram um nível de exigência de conhecimento de conceitos computacionais (técnicas algoritmicas e de estruturação de dados) e também de conhecimentos estatísticos (probabilidade geral e teoria de filas) não usualmente experimentado durante um curso padrão de graduação em Estatística. Através do desenvolvimento de funções específicas que calculam métricas associadas às filas, como a probabilidade de bloqueio após absorver um determinado número de usuários no sistema (`calc_PK`), a probabilidade de ausência de usuários (`calc_P0`) e o tempo médio de espera (`calc_W`), foi possível quantificar diversos aspectos críticos que impactam o desempenho geral de tais sistemas.

As funções aqui desenvolvidas permitem que administradores de sistemas, engenheiros e tomadores de decisão avaliem a eficácia operacional de suas respectivas infraestruturas. Isso faz do algoritmo apresentado, uma ferramenta aplicável em diversos cenários práticos de grande interesse.

4.1 Implicações e Limitações

O presente estudo, apesar de seus méritos na otimização de sistemas de filas do tipo $M/M/c/k$, apresenta um conjunto de desafios e limitações que merecem atenção. Primeiramente, o código carece de critérios de parada mais robustos. Na ausência de uma taxa de atendimento geral θ aceitável durante a execução da heurística *Simulated Annealing*, o algoritmo pode entrar em um *loop* infinito. Este problema é contornável, mas carece de novas inserções algoritmicas no mecanismo até aqui implementado. Similarmente, a ausência de um critério de parada para a solução inicial pode resultar em comportamento indesejado se as condições de *buffer*, servidores e θ restrito forem inatingíveis.

Além disso, o tratamento de dados de entrada é inflexível. Se um usuário inserir um valor que diverge do formato esperado, o sistema não possui mecanismos para adaptar-se ou corrigir o erro. O modelo também poderia beneficiar-se de algoritmos de vizinhança mais sofisticados, possivelmente com um banco de dados para rastrear vizinhos já visitados e evitar redundâncias.

Do ponto de vista teórico, o modelo possui um escopo restrito, limitando-se às redes de filas acíclicas do tipo $M/M/c/k$. Isso exclui a possibilidade de aplicação em

redes com estratégias de retrabalho, por exemplo. A implementação atual também não contempla filas sem limitações de *buffers*, ou seja, filas $M/M/c$. Além disso, mecanismos mais sofisticados para correções para filas não markovianas não estão ainda atendidos.

4.1.1 Propostas para Trabalhos Futuros

1. **Implementação de critérios de parada:** Introduzir mecanismos que previnam *loops* infinitos na heurística *Simulated Annealing* e na fase de produção de soluções iniciais.

2. **Melhorias no tratamento de dados de entrada:** Desenvolver uma interface de usuário mais robusta e flexível para garantir que dados inseridos incorretamente possam ser corrigidos ou adaptados.

3. **Otimização da vizinhança:** Investigar e implementar algoritmos de vizinhança mais eficazes, incorporando um banco de dados para mapear vizinhos já visitados.

4. **Avaliações para problemas de otimização multiobjetivo:** Adaptar o algoritmo para considerar múltiplos objetivos de otimização, isso permitirá uma abordagem mais holística à gestão das redes de filas.

5. **Extensão para outros modelos de fila:** Expandir o algoritmo para acomodar diferentes tipos de filas, com a inclusão, mas não limitação à filas $M/M/c$, $M/G/c$, $G/M/c$, $G/G/c$, $G[x]/G/c$, $M/G/c/k$, $G/M/c/k$, $G/G/c/k$ e $G[x]/G/c/k$.

6. **Modelagem temporal avançada:** Incorporar elementos de séries temporais para simular variações nas taxas de chegada e serviço ao longo do tempo, ou seja, modelos com taxas dependentes de tempo.

7. **Integração com outros tipos de fila:** Ampliar o modelo para incluir outros tipos de fila, como filas com esquemas de prioridade, redes de filas com abandonos, esquemas de retrabalho.

8. **Otimização baseada em restrições:** Introduzir complexidade adicional ao permitir que os usuários definam restrições específicas, como o custo máximo operacional ou o tempo médio de serviço desejado, e encontrar a configuração ótima que atenda a essas restrições.

9. **Aplicação em grandes escalas:** Testar o desempenho do otimizador em sistemas de fila de grande escala, como redes de telecomunicações, hospitais e aeroportos.

10. **Integração com aprendizado de máquina:** Utilizar modelos de aprendizado de máquina para prever cenários futuros baseados em dados históricos, o que poderia melhorar a eficácia das decisões de otimização.

Por fim, o trabalho atual estabelece uma base razoável para a análise e otimização de redes de filas. Para torná-lo uma ferramenta completa e abrangente em diversas aplicações práticas, é crucial abordar as limitações e expandir o escopo do modelo, conforme delineado nas propostas para trabalhos futuros.

Referências

[1] Ahmed, N. U. e Ouyang, X. H.: *Suboptimal RED feedback control for buffered TCP flow dynamics in computer network*. Mathematical Problems in Engineering, 2007(Article ID 54683):17 pages, 2007, ISSN 1024-123X. Citado 2 vezes nas páginas 1 e 8.

[2] Alves, F. S. Q., Yehia, H. C., Pedrosa, L. A. C., Cruz, F. R. B. e Kerbache, L.: *Upper bounds on performance measures of heterogeneous M/M/c queues*. Mathematical Problems in Engineering, 2011(Article ID 702834):18 pages, 2011. Citado na página 7.

[3] Bruin, A. M., Rossum, A. C. v., Visser, M. C. e Koole, G. M.: *Modeling the emergency cardiac in-patient flow: An application of queuing theory*. Health Care Management Science, 10(2):125–137, 2007. Citado na página 7.

[4] Canadilla, P.: *queueing: Analysis of Queueing Networks and Models*, 2019. `https://CRAN.R-project.org/package=queueing`, R package version 0.2.12. Citado na página 16.

[5] Černỳ, V.: *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*. Journal of optimization theory and applications, 45:41–51, 1985. Citado na página 12.

[6] Chaudhuri, K., Kothari, A., Pendavingh, R., Swaminathan, R., Tarjan, R. e Zhou, Y.: *Server allocation algorithms for tiered systems*. Algorithmica, 48(2):129–146, 2007. Citado na página 8.

[7] Chen, J., Hu, C. e Ji, Z.: *An improved ARED algorithm for congestion control of network transmission*. Mathematical Problems in Engineering, 2010(Article ID 329035):17 pages, 2010, ISSN 1024-123X. Citado 2 vezes nas páginas 1 e 8.

[8] Cruz, F. R. B., Duarte, A. R. e Woensel, T. V.: *Buffer allocation in general single-server queueing networks*. Computers & Operations Research, 35(11):3581–3598, 2008. Citado 3 vezes nas páginas 1, 5 e 8.

[9] Cruz, F. R. B., Smith, J. M. e Medeiros, R. O.: *An M/G/C/C state dependent network simulation model*. Computers & Operations Research, 32(4):919–941, 2005. Citado na página 8.

[10] Cruz, F. R. B., Woensel, T. V., Smith, J. M. e Lieckens, K.: *On the system optimum of traffic assignment in M/G/c/c state-dependent queueing networks*. European Journal of Operational Research, 201(1):183–193, 2010. Citado na página 8.

[11] Csardi, G. e Nepusz, T.: *The igraph software package for complex network research*. InterJournal, Complex Systems:1695, 2006. `https://igraph.org`. Citado na página 16.

[12] Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T.: *A fast and elitist multiobjective genetic algorithm: Nsga-ii*. IEEE transactions on evolutionary computation, 6(2):182–197, 2002. Citado na página 2.

[13] Dimitriou, I. e Langaris, C.: *A repairable queueing model with two-phase service, start-up times and retrial customers*. Computers and Operations Research, 37(7):1181–1190, 2010. Citado na página 8.

[14] Duarte, A. R.: *The Server Allocation Problem for markovian queueing networks*. International Journal of Services and Operations Management, (to appear), 2022. `http://dx.doi.org/10.1504/IJSOM.2022.10047177`. Citado 2 vezes nas páginas 2 and 5.

[15] Hillier, F. S.: *Economic models for industrial waiting line problems*. Management Science, 10(1):119–130, 1963. Citado na página 5.

[16] Inzillo, V., De Rango, F. e Quintana, A. A.: *A self clocked fair queuing MAC approach limiting deafness and round robin issues in directional MANET*. Em *2019 Wireless Days (WD)*, pp. 1–6. IEEE, 2019. Citado 2 vezes nas páginas 1 e 8.

[17] Kendall, D. G.: *Stochastic processes occurring in the theory of queues and their analysis by the method of embedded Markov chains*. Annals Mathematical Statistics, 24:338–354, 1953. Citado na página 6.

[18] Kirkpatrick, S., Gelatt Jr, C. D. e Vecchi, M. P.: *Optimization by simulated annealing*. science, 220(4598):671–680, 1983. Citado na página 12.

[19] Martins, H. S. R., Cruz, F. R. B., Duarte, A. R. e Oliveira, F. L. P.: *Modeling and optimization of buffers and servers in finite queueing networks*. OPSEARCH, 56(1):123–150, 2019. Citado na página 5.

[20] Menasce, D. A.: *QoS issues in web services*. IEEE Internet Computing, 6(6):72–75, 2002. Citado na página 8.

[21] Osorio, C. e Bierlaire, M.: *An analytic finite capacity queueing network model capturing the propagation of congestion and blocking*. European Journal of Operational Research, 196(3):996–1007, 2009. Citado na página 8.

[22] Pedersen, T. L.: *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks*, 2022. `https://CRAN.R-project.org/package=ggraph`, R package version 2.1.0. Citado na página 16.

[23] Pedersen, T. L.: *tidygraph: A Tidy API for Graph Manipulation*, 2023. `https://CRAN.R-project.org/package=tidygraph`, R package version 1.2.3. Citado na página 16.

[24] Smith, J. M. e Cruz, F. R. B.: *The buffer allocation problem for general finite buffer queueing networks*. IIE Transactions, 37(4):343–365, 2005. Citado 3 vezes nas páginas 13, 9 e 10.

[25] Smith, J. M., Cruz, F. R. B. e Woensel, T. V.: *Topological network design of general, finite, multi-server queueing networks*. European Journal of Operational Research, 201(2):427–441, 2010. Citado na página 7.

[26] Souza, G. L.: *Uma nova formulação para otimização multi-objetivo em redes de filas finitas gerais e com único servidor*. Tese de Mestrado, Universidade Federal de Ouro Preto, Ouro Preto, 2020. Citado na página 6.

[27] Souza, G. L., Duarte, A. R., Moreira, G. J. P. e Cruz, F. R. B.: *A novel formulation for multi-objective optimization of general finite single-server queueing networks*. Em *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2020. Citado na página 2.

[28] Wickham, H.: *stringr: Simple, Consistent Wrappers for Common String Operations*, 2022. `https://CRAN.R-project.org/package=stringr`, R package version 1.5.0. Citado na página 16.

[29] Yang, X. S.: *Engineering optimization: An introduction with metaheuristic applications*. Wiley Publishing, 1st ed., 2010, ISBN 0470582464, 9780470582466. Citado 2 vezes nas páginas 13 e 8.

Anexos

ANEXO A – Implementação Geral

```
pacman::p_load("stringr","ggraph", "tidygraph", "igraph","queueing")

##-----DEFININDO FUNCOES PK, P0 E W e ETL#####################

#definindo a funcao PK
calc_PK=function(lambda,mu,c,K){
  r=lambda/mu  #intensidade do servidor
  rho=r/c  #intensidade do tráfego
  sum_j=0
  if(c>1){
     for(j in 1:c){
        sum_j=sum_j+((r^j)/factorial(j))
     }
  }else{
     sum_j=r
  }
  if(rho==1){# caso rho = 1
     den_j=1+sum_j+(r^c)*(K-c+1)/factorial(c)
     num_j=(r^c)*(rho^(K-c))/factorial(c)
     PK=num_j/den_j #calculo PK
  }else{# caso rho <> 1
     den_j2=1+sum_j+(r^c)*rho*(1-(rho^(K-c)))/(factorial(c)*(1-rho))
     num_j2=(r^c)*(rho^(K-c))/factorial(c)
     PK=num_j2/den_j2 #calculo PK
  }
  return(PK)
}

# Definindo a funcao para calcular P0
calc_P0=function(lambda,mu,c,K){
  r=lambda/mu  #intensidade do servidor
  rho=r/c  #intensidade do tráfego
  sum_j=0
  if(c>1){
     for(j in 1:c){
```

```
        sum_j=sum_j+((r^j)/factorial(j))
     }
  }else{
     sum_j=r
  }
  if(rho==1){# caso rho = 1
     den_j=1+sum_j+(r^c)*(K-c+1)/factorial(c)
     P0=1/den_j #calculo PK
  }else{# caso rho <> 1
     den_j2=1+sum_j+(r^c)*rho*(1-(rho^(K-c)))/(factorial(c)*(1-rho))
     P0=1/den_j2 #calculo PK
  }
  return(P0)
}

# Definindo a funcao para calcular W
calc_W=function(lambda,mu,c,K,PK,P0){
  r=lambda/mu  #intensidade do servidor
  rho=r/c  #intensidade do tráfego
  if(rho==1){
     num_1=2*r*(1-PK)*factorial(c)+(r^c)*P0*(K-c+1)*(K-c)
     den_1=2*lambda*(1-PK)*factorial(c)
     W=num_1/den_1
  }else{
    num_2=(r^c)*P0*(1-(K-c+1)*rho^(K-c)+(K-c)*rho^(K-c+1))
    den_2=lambda*(1-PK)*factorial(c)*(1-rho)^2
    W=(1/mu)+num_2/den_2
  }
  return(W)
}

# Definindo a funçao ETL do arquivo e capturando todos os parametros
ETL=function(caminho){
  #Ler o arquivo de texto obtidos de caminho
  linhas = readLines(caminho)
  #leitura por este modo, trouxe \t nos espaços
  data = str_replace_all(linhas, "\t"," ")  %>%
  #limpeza dos \t e demais formatacoes
    str_squish() ## trim dos espaços em branco
```

```
# Criando uma lista para armazenar os dados lidos
parametros = list()

# Percorrendo as linhas do arquivo
for (linha in data) {
  # Verificando se a linha contém um número
  if (grepl("^[0-9]", linha)) {
    #se sim, adiciona a linha à lista
    parametros = c(parametros,linha)
  }
}
##EXTRACAO DOS PARAMETROS
#NUMBER OF QUEUES, NQ
NQ <<- as.numeric(parametros[1])
#NUMBER OF ARCS, NARCS
NARCS <<- as.numeric(parametros[2])

#########EXTRACAO DA MATRIZ 1
matriz_base_nodes = list()

for (i in 1:NARCS) {
  matriz_base_nodes = c(matriz_base_nodes, parametros[2+i])
  #comeCar da 3 linha, onde inicia a matriz de nodes
}

#tratamento de espacos, transformacao
#de lista->matriz e de texto->numerico
matriz_nodes <- do.call(
  rbind, lapply(
    strsplit(unlist(
      str_replace_all(matriz_base_nodes,
                      " ",
                      ",")
    ),","),
    as.numeric)
)
 #linha final para continuar o ETL em parametros
final = 2+length(matriz_base_nodes)
```

```
#NUMBER OF SOURCE NODES, NSNOD
NSNODE <<- as.numeric(parametros[final+1])

#########EXTRACAO DA MATRIZ 2
matriz_base_arrival = list()

for (i in 1:NSNODE) {
  matriz_base_arrival = c(matriz_base_arrival, parametros[final+1+i])
  #comecando da posicao de NSNODE+1
}
 #linha em que termina a matriz 2
final2 = final+1+length(matriz_base_arrival)

#tratamento de espacos, transformacao de
#lista->matriz e de texto->numerico
matriz_arrival <<- do.call(
  rbind, lapply(
    strsplit(unlist(
      str_replace_all(matriz_base_arrival,
                      " ",
                      ",")
    ),","),
    as.numeric)
)

#NUMBER OF SINK NODES, NEND || NUMBER OF END NODES, NEND
NEND <<- as.numeric(parametros[final2+1])

#INSINK NODES, ENDD ||  END NODES, ENDD
linhaENDD = as.character(parametros[final2+2])

ENDD <<- as.numeric(strsplit(linhaENDD, " ")[[1]])

###MATRIZ 3##
matriz_base_service = list()

for (i in 1:NQ) {
  matriz_base_service = c(matriz_base_service, parametros[final2+2+i])
```

```
    #comecando da posicao de ENDD+1
  }

  matriz_service <<- do.call(
    rbind, lapply(
      strsplit(unlist(
        str_replace_all(matriz_base_service,
                        " ",
                        ",")
      ),","),
      as.numeric)
  )

  #PRINT DOS PARAMETROS

  cat("#NUMBER OF QUEUES, NQ\n", NQ, "\n")
  cat("#NUMBER OF ARCS, NARCS\n", NARCS, "\n")

  cat("#ARC STARTING & ENDING NODES,
  AND ROUTING PROBABILITIES, NS(I), NF(I), RP(I)\n")
  print(matriz_nodes)

  cat("#NUMBER OF SOURCE NODES, NSNOD\n", NSNODE, "\n")

  cat("#SOURCE NODES AND ARRIVAL RATE, SOUR, SLAM(I)\n")
  print(matriz_arrival)

  cat("#NUMBER OF SINK NODES, NEND ||
  NUMBER OF END NODES, NEND\n", NEND, "\n")
  cat("#INSINK NODES, ENDD ||
  END NODES, ENDD\n", ENDD, "\n")

  cat("#SERVICE RATES, COEFF VARIATION,
  AND TOT CAPACITY, RATE(I), AS(I), RW(I)\n")
  print(matriz_service)


#-----MONTANDO O GRAFO
#gerando o grafo que modelam as filas
```

```
grafo <<- graph_from_edgelist(matriz_nodes[,1:2], directed = TRUE)
E(grafo)$weight <<- matriz_nodes[,3]
#adicionando peso as arestas (probabilidade)

#----- lotando o grafo para visualização
#print funcional
E(grafo)$label <<- paste("P (",E(grafo)$weight,")")
tkplot(grafo, vertex.label.color = "black", vertex.label = V(grafo)$label,
       edge.label.color = "black", edge.label=E(grafo)$label,
       vertex.color = "yellow", edge.color = "black", edge.label.cex = 1)

#Função para encontrar todos os caminhos
# Para cada nó de origem
for (origem in matriz_arrival[, 1]) {
  # Para cada nó de destino
  for (destino in ENDD) {
    # Calcula todos os caminhos
    caminhos <<- all_simple_paths(grafo, from = origem, to = destino)
    caminhos <<- do.call(rbind, lapply(caminhos, as_ids))
    #junta as listas em uma matriz de caminhos
    print(paste("Todos os caminhos de", origem, "para", destino, ":"))
    print(caminhos)
    }
  }
#---- Definindo os atributos dos vértices (filas)
V(grafo)$mu <<- matriz_service[,1]
#taxa de saida de cada servidor da fila Mi
V(grafo)$W <<- 0 #Tempo de espera médio do sistema
V(grafo)$PK <<- 0 #probabilidade de bloqueio
V(grafo)$name <<- as_ids(V(grafo)) #nome do vertice poelo ID
V(grafo)$theta <<- 0
# Primeiro, inicializamos ‘theta‘ para todos os nós como 0
V(grafo)$IS_SN <<- FALSE
# Inicialmente, atribui-se FALSE para IS_SOURCENODE
for (i in 1:length(V(grafo))) {
# Agora percorre todos os vértices do grafo
  # Se o nome do vértice estiver
  # presente na primeira coluna da matriz_arrival
  if (V(grafo)[i]$name %in% matriz_arrival[, 1]) {
```

```
    # Atribui TRUE para IS_SN
    V(grafo)[i]$IS_SN <<- TRUE
  }
}
V(grafo)$IS_EN <<- FALSE # Inicialmente, atribui-se FALSE para IS_EN
for (i in 1:length(V(grafo))) {
# Agora percorre todos os vértices do grafo
  # Se o nome do vértice estiver presente na variável ENDD
  if (V(grafo)[i]$name %in% ENDD) {
    # Atribui TRUE para IS_EN
    V(grafo)[i]$IS_EN <<- TRUE
  }
}
#lambda
V(grafo)$lambda <<- 0
for (i in 1:length(V(grafo))) { # Percorre todos os vértices do grafo
  if (V(grafo)[i]$IS_SN) { # Se o vértice for um nó de origem
    # Busca o valor correspondente de lambda na matriz_arrival
    V(grafo)[i]$lambda <<- matriz_arrival[matriz_arrival[, 1]
    == as.numeric(V(grafo)[i]$name), 2]
  }
}
}

# Definindo a funcao para calcular Theta
calc_Theta=function(grafo,matriz_arrival){
  for (i in 1:length(V(grafo))){

    # Recupera as taxas de serviço e
    # a quantidade de servidores do vértice
    mu=V(grafo)[i]$mu  # taxa de serviço
    c=V(grafo)[i]$servidores  # número de servidores
    K=V(grafo)[i]$buffer+c  # Tamanho da fila (buffer + servidores)

    if(V(grafo)[i]$IS_SN){  # nó de origem
      lambda=matriz_arrival[matriz_arrival[,1]==V(grafo)[i]$name,2]
      # taxa de chegada no nó
    }else{  # nós subsequentes
      # Recuperamos todas as arestas que chegam ao nó i
```

```
    edges_in=incident(grafo,V(grafo)[i],mode="in")
    neighbors_in=neighbors(grafo,V(grafo)[i],mode="in")
    lambda=0
    for(j in 1:length(neighbors_in)){
      lambda=lambda
      +(V(grafo)[neighbors_in[j]]$theta*E(grafo)[edges_in[j]]$weight)
    }
  }

  # Atualizamos o lambda para o nó i
  V(grafo)[i]$lambda=lambda



  # Calcula a probabilidade de bloqueio e atribui ao vértice
  V(grafo)[i]$PK=calc_PK(lambda,mu,c,K)

  # Calcula P0 e atribui ao vértice
  V(grafo)[i]$P0=calc_P0(lambda,mu,c,K)

  # Calcula o tempo médio de fila e atribui ao vértice
  V(grafo)[i]$W=calc_W(lambda,mu,c,K,V(grafo)[i]$PK,V(grafo)[i]$P0)

  # Calculamos theta para o nó i a
  #partir do lambda e da probabilidade de bloqueio
  V(grafo)[i]$theta=V(grafo)[i]$lambda*(1-V(grafo)[i]$PK)
}

# Inicializa Theta como 0
theta_final=0
for (i in 1:length(V(grafo))) {
  # Verifica se o vértice é um nó final
  if(V(grafo)[i]$IS_EN==TRUE){
    # Adiciona o valor de theta do nó final ao Theta
    theta_final=theta_final+V(grafo)[i]$theta
  }
}
grafo <<- grafo
#theta_final <<- theta_final
return(theta_final)
```

```
  #return(list(grafo = grafo, theta_final = theta_final))
}

# Definindo a funcao para calcular W Total (do circuito)
calc_TotalW = function(grafo, caminhos) {

  # Inicializa Wtotal
  Wtotal = 0
  # Percorre todos os caminhos
  for(i in 1:length(caminhos[,1])){
    # Inicializa o peso do caminho atual
    peso_caminho = 1
    # Percorre todos os nós no caminho atual
    for(j in 1:(length(caminhos[i,])-1)){
      # Identifica a aresta que liga o nó atual
      #ao próximo nó no caminho
      edge=get.edge.ids(grafo,c(V(grafo)[caminhos[i,j]],
      V(grafo)[caminhos[i,j+1]]))
      # Atualiza o peso do caminho
      peso_caminho=peso_caminho*E(grafo)[edge]$weight
    }
    # Soma o produto do peso do caminho pelo
    #somatório dos W dos nós no caminho ao Wtotal
    Wtotal=Wtotal+peso_caminho*
    sum(V(grafo)[caminhos[i,1:(length(caminhos[i,])-1)]]$W)
  }
  return(Wtotal)
}

# Definindo a funcao para calcular
#Solucoes iniciais pseudoaleatorias
initial_Solutions = function(grafo,matriz_arrival,n_starts,theta_restrict,
correct_buffers,correct_servers){
  Tot_buffers=correct_buffers*length(V(grafo))
  Tot_servers=correct_servers*length(V(grafo))
  initial_solutions=NULL
  for(i in 1:n_starts){
    buffers=rep(0,length(V(grafo)))
    servers=rep(0,length(V(grafo)))
```

```
    a=runif(1,-0.5,0.5)
    Tot_buffers=floor((1+a)*Tot_buffers)
    a=runif(1,-0.5,0.5)
    Tot_servers=floor((1+a)*Tot_servers)
    stop=0
    while(stop==0){
      buffers=rmultinom(1,size=Tot_buffers,
      prob=rep(1/length(V(grafo)),length(V(grafo))))
      while (any(servers==0)){
        servers=rmultinom(1,size=Tot_servers,
        prob=rep(1/length(V(grafo)),length(V(grafo))))
      }
      #calcula theta
      V(grafo)$servidores=servers
      V(grafo)$buffer=buffers
      theta=calc_Theta(grafo,matriz_arrival)
      if(theta>=theta_restrict) stop=1 #90% de lambda de entrada
    }
    if(i==1){
      initial_solutions=c(buffers,servers)
    }else{
      initial_solutions=cbind(initial_solutions,c(buffers,servers))
    }
    Tot_buffers=correct_buffers*length(V(grafo))
    Tot_servers=correct_servers*length(V(grafo))
  }
  print(theta)
  return(initial_solutions)
}

##-----INPUT DOS DADOS ####################
ETL(file.choose())

##-----SOLUCOES INICIAIS ####################

qtd_Solini = 3
theta_restrict = 0.8
correcao_Buffers = 10
correcao_Servidores = 10
```

```
sol_ini=initial_Solutions(
  grafo,
  matriz_arrival,
  qtd_Solini,
  theta_restrict*sum(matriz_arrival[,2]),
  correcao_Buffers,
  correcao_Servidores
)

##----- SIMULATED ANNEALING ####################

#custo
calc_Cost=function(alpha,solution,NQ){
  cost=alpha*sum(solution[1:NQ])+(1-alpha)*sum(solution[(NQ+1):(NQ+NQ)])
  return(cost)
}

#sa function
SA=function(grafo1,sol_ini,NQ,theta_restrict,cont_max){
# parametros
  alpha=0.1
  p_viz_1=0.1
  p_viz_2=0.2
  p_viz_3=0.2
  p_viz_4=0.1
  p_viz_5=0.2
  p_viz_6=1-p_viz_1-p_viz_2-p_viz_3-p_viz_4-p_viz_5
  sol_max=sol_ini
  sol_cur=sol_ini
  F_max=calc_Cost(alpha,sol_max,NQ) # Inicializa F_max aqui
  cont=1
  stop=0
  while(stop==0){
    u_viz=runif(1)#
    print(paste("SA rodada: ",cont))

    ##Critério de vizinhança 1 - Troca de Buffer ####
    if(u_viz<p_viz_1) {
```

```
sol_new=sol_cur
index=sample(c(1:NQ),size=2,replace=FALSE)

#swap
aux=sol_new[index[1]]
sol_new[index[1]]=sol_new[index[2]]
sol_new[index[2]]=aux

# Atualizando o grafo
for(i in (NQ+1):(2*NQ)){
   V(grafo1)[(i-NQ)]$servidores=sol_new[i]
   V(grafo1)[(i-NQ)]$buffer=sol_new[(i-NQ)]
}

# Calculando custos
F_new=calc_Cost(alpha,sol_new,NQ)
F_cur=calc_Cost(alpha,sol_cur,NQ)

# Calculando tempo
W_new=calc_TotalW(grafo,caminhos)
W_cur=calc_TotalW(grafo1,caminhos)

# Calculo de Theta Novo
theta_new=calc_Theta(grafo1,matriz_arrival)

# Atualização dos indicadores
if(theta_new>=theta_restrict){
  if(W_new<=W_cur){
    sol_cur=sol_new
    cont=cont+1
    if(F_new<=F_max) {
      sol_max=sol_new
      F_max=F_new
    }
  }else{
    u=runif(1)
    if(u<=cont^(W_cur-W_new)){
      sol_cur=sol_new
      cont=cont+1
```

```
      }
    }
  }
}

##Critério de vizinhança 2 - +1 Buffer ####
if(u_viz>=p_viz_1 && u_viz<(p_viz_1+p_viz_2)){
  sol_new=sol_cur
  index=sample(c(1:NQ),size=1,replace=FALSE)
  sol_new[index]=sol_new[index]+1

  # Calculando custos
  F_new=calc_Cost(alpha,sol_new,NQ)
  F_cur=calc_Cost(alpha,sol_cur,NQ)

  # Atualizando o grafo
  for(i in (NQ+1):(2*NQ)){
     V(grafo1)[(i-NQ)]$servidores=sol_new[i]
     V(grafo1)[(i-NQ)]$buffer=sol_new[(i-NQ)]
  }

  # Calculo de Theta Novo
  theta_new=calc_Theta(grafo1,matriz_arrival)

  # Atualização dos indicadores
  if(theta_new>=theta_restrict){
    if(F_new<=F_cur){
      sol_cur=sol_new
      cont=cont+1
      if(F_new<=F_max){
        sol_max=sol_new
        F_max=F_new
      }
    }else{
      u=runif(1)
      if(u<=cont^(F_cur-F_new)){
        sol_cur=sol_new
        cont=cont+1
      }
```

```
    }
  }
}

##Critério de vizinhança 3 - -1 Buffer ####
if(u_viz>=(p_viz_1+p_viz_2) && u_viz<(p_viz_1+p_viz_2+p_viz_3)){
  sol_new=sol_cur

  aux=0
  while(aux==0){
    index=sample(c(1:NQ),size=1,replace=FALSE)
    aux=sol_new[index]
  }
  sol_new[index]=sol_new[index]-1

  # Calculando custos
  F_new=calc_Cost(alpha, sol_new, NQ)
  F_cur=calc_Cost(alpha, sol_cur, NQ)

  # Atualizando o grafo
  for(i in (NQ+1):(2*NQ)){
     V(grafo1)[(i-NQ)]$servidores=sol_new[i]
     V(grafo1)[(i-NQ)]$buffer=sol_new[(i-NQ)]
  }

  # Calculo de Theta Novo
  theta_new=calc_Theta(grafo1,matriz_arrival)

  # Atualização dos indicadores
  if(theta_new>=theta_restrict){
    if(F_new<=F_cur){
      sol_cur=sol_new
      cont=cont+1
      if(F_new<=F_max){
        sol_max=sol_new
        F_max=F_new
      }
    }else{
      u=runif(1)
```

```
      if(u<=cont^(F_cur-F_new)){
        sol_cur=sol_new
        cont=cont+1
      }
    }
  }
}

##Critério de vizinhança 4 - Troca de servidores ####
if(u_viz>=(p_viz_1+p_viz_2+p_viz_3)
&& u_viz<(p_viz_1+p_viz_2+p_viz_3+p_viz_4)){
  sol_new = sol_cur
  index=sample(c((NQ+1):(NQ+NQ)),size=2,replace=FALSE)

  #swap
  aux=sol_new[index[1]]
  sol_new[index[1]]=sol_new[index[2]]
  sol_new[index[2]]=aux

  # Atualizando o grafo
  for(i in (NQ+1):(2*NQ)){
     V(grafo1)[(i-NQ)]$servidores=sol_new[i]
     V(grafo1)[(i-NQ)]$buffer=sol_new[(i-NQ)]
  }

  # Calculando custos
  F_new=calc_Cost(alpha,sol_new,NQ)
  F_cur=calc_Cost(alpha,sol_cur,NQ)

  W_new=calc_TotalW(grafo1,caminhos)
  W_cur=calc_TotalW(grafo1,caminhos)

  # Calculo de Theta Novo
  theta_new=calc_Theta(grafo1,matriz_arrival)

  # Atualização dos indicadores
  if(theta_new>=theta_restrict){
    if(W_new<=W_cur) {
      sol_cur=sol_new
```

```
      cont=cont+1
      if(F_new<=F_max) {
        sol_max=sol_new
        F_max=F_new
      }
    }else{
      u=runif(1)
      if(u<=cont^(W_cur-W_new)){
        sol_cur=sol_new
        cont=cont+1
      }
    }
  }
}


##Critério de vizinhança 5 - +1 servidor ####
if(u_viz>=(p_viz_1+p_viz_2+p_viz_3+p_viz_4)
&& u_viz<(p_viz_1+p_viz_2+p_viz_3+p_viz_4+p_viz_5)){
  sol_new = sol_cur

  index=sample(c((NQ+1):(NQ+NQ)),size=1,replace=FALSE)
  sol_new[index]=sol_new[index]+1

  # Calculando custos
  F_new = calc_Cost(alpha, sol_new, NQ)
  F_cur = calc_Cost(alpha, sol_cur, NQ)

  # Atualizando o grafo
  for(i in (NQ+1):(2*NQ)){
     V(grafo1)[(i-NQ)]$servidores=sol_new[i]
     V(grafo1)[(i-NQ)]$buffer=sol_new[(i-NQ)]
  }

  # Calculando custos
  F_new=calc_Cost(alpha,sol_new,NQ)
  F_cur=calc_Cost(alpha,sol_cur,NQ)

  W_new=calc_TotalW(grafo1,caminhos)
```

```
  W_cur=calc_TotalW(grafo1,caminhos)

  # Calculo de Theta Novo
  theta_new=calc_Theta(grafo1,matriz_arrival)

  # Atualização dos indicadores
  if(theta_new>=theta_restrict){
    if(F_new<=F_cur) {
      sol_cur=sol_new
      cont=cont+1
      if(F_new<=F_max) {
        sol_max=sol_new
        F_max=F_new
      }
    }else{
      u=runif(1)
      if(u<=cont^(F_cur-F_new)){
        sol_cur=sol_new
        cont=cont+1
      }
    }
  }
}

##Critério de vizinhança 6 - -1 servidor ####
if(u_viz>=(p_viz_1+p_viz_2+p_viz_3+p_viz_4+p_viz_5)){
  sol_new=sol_cur

  aux=0
  while(aux<=1){
    index=sample(c((NQ+1):(NQ+NQ)),size=1,replace=FALSE)
    aux=sol_new[index]
  }
  sol_new[index]=sol_new[index]-1

  # Calculando custos
  F_new=calc_Cost(alpha, sol_new, NQ)
  F_cur=calc_Cost(alpha, sol_cur, NQ)
```

```
      # Atualizando o grafo
      for(i in (NQ+1):(2*NQ)){
         V(grafo1)[(i-NQ)]$servidores=sol_new[i]
         V(grafo1)[(i-NQ)]$buffer=sol_new[(i-NQ)]
      }

      # Calculo de Theta Novo
      theta_new=calc_Theta(grafo1,matriz_arrival)

      # Atualização dos indicadores
      if(theta_new>=theta_restrict) {
        if(F_new<=F_cur) {
          sol_cur=sol_new
          cont=cont+1
          if(F_new<=F_max) {
            sol_max=sol_new
            F_max=F_new
          }
        }else{
          u=runif(1)
          if(u<=cont^(F_cur-F_new)) {
            sol_cur=sol_new
            cont=cont+1
          }
        }
      }
    }

    ## Fim do S.A. ####
    if(cont == cont_max) {
      print("Condição de parada atingida.")
      stop = 1
    }
  }
  return(sol_max)
}

##----- MAIN ###################################
sol_fim = matrix(0,nrow=length(sol_ini[,1]),ncol=length(sol_ini[1,]))
```

```
theta_restrict=0.8*sum(matriz_arrival[,2])
for(i in 1:length(sol_ini[1,])){
  print(paste("Solução inicial: ",i))
  sol_fim[,i]=SA(grafo,sol_ini[,i],NQ,theta_restrict, 100)
}


#########Verificação de qualidade
alpha=0.1
NQ=16
for(j in 1:length(sol_ini[1,])){
   A=calc_Cost(alpha,sol_ini[,j],NQ)
   B=calc_Cost(alpha,sol_fim[,j],NQ)
   print(paste("Solução: ",j))
   print(paste("Custo inicial: ",A,"Custo final: ",B))
   grafo2=grafo
      for(i in (NQ+1):(2*NQ)){
         V(grafo2)[(i-NQ)]$servidores=sol_ini[i]
         V(grafo2)[(i-NQ)]$buffer=sol_ini[(i-NQ)]
      }
   A=calc_Theta(grafo2,matriz_arrival)
      for(i in (NQ+1):(2*NQ)){
         V(grafo2)[(i-NQ)]$servidores=sol_fim[i]
         V(grafo2)[(i-NQ)]$buffer=sol_fim[(i-NQ)]
      }
   B=calc_Theta(grafo2,matriz_arrival)
   print(paste("Theta inicial: ",A,"Theta final: ",B))
}
```