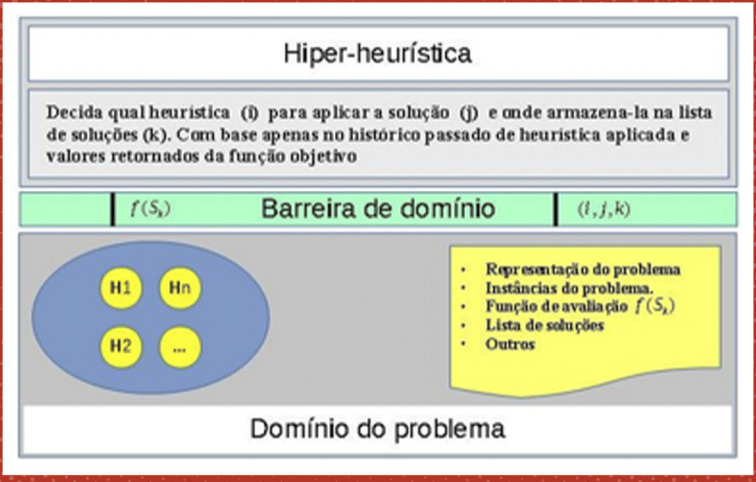
Antonio Clerton Santana de Araujo

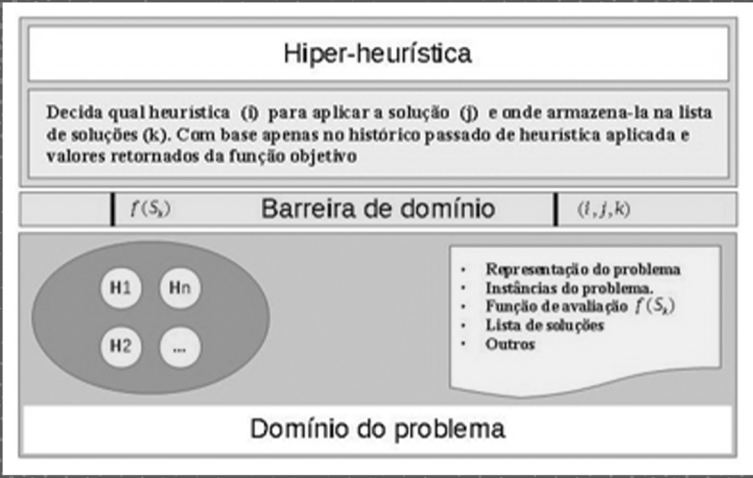
Uma abordagem hiper-heurística para problemas de otimização



Atena Editora
Ano 2023

Antonio Clerton Santana de Araujo

Uma abordagem hiper-heurística para problemas de otimização



Atena Editora
Ano 2023

CIÊNCIAS EXATAS E DA TERRA E ENGENHARIAS

Editora chefe
Profª Drª Antonella Carvalho de Oliveira
Editora executiva
Natalia Oliveira
Assistente editorial
Flávia Roberta Barão
Bibliotecária
Janaina Ramos
Projeto gráfico
Camila Alves de Cremo
Luiza Alves Batista
Imagens da capa
iStock
Edição de arte
Luiza Alves Batista

CIÊNCIAS EXATAS E DA TERRA E ENGENHARIAS

Uma abordagem hiper-heurística para problemas de otimização

Diagramação: Camila Alves de Cremo
Correção: Maiara Ferreira
Indexação: Amanda Kelly da Costa Veiga
Revisão: O autor
Autor: Antonio Clerton Santana de Araujo

Dados Internacionais de Catalogação na Publicação (CIP)

A663 Araujo, Antonio Clerton Santana de
Uma abordagem hiper-heurística para problemas de otimização / Antonio Clerton Santana de Araujo. – Ponta Grossa - PR: Atena, 2023.

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui bibliografia
ISBN 978-65-258-1279-3
DOI: https://doi.org/10.22533/at.ed.793232003

1. Otimização combinatória. 2. Programação heurística. I. Araujo, Antonio Clerton Santana de. II. Título.
CDD 004.36

Elaborado por Bibliotecária Janaina Ramos – CRB-8/9166

Atena Editora
Ponta Grossa – Paraná – Brasil
Telefone: +55 (42) 3323-5493
www.atenaeditora.com.br
contato@atenaeditora.com.br

DECLARAÇÃO DO AUTOR

O autor desta obra: 1. Atesta não possuir qualquer interesse comercial que constitua um conflito de interesses em relação ao conteúdo publicado; 2. Declara que participou ativamente da construção dos respectivos manuscritos, preferencialmente na: a) Concepção do estudo, e/ou aquisição de dados, e/ou análise e interpretação de dados; b) Elaboração do artigo ou revisão com vistas a tornar o material intelectualmente relevante; c) Aprovação final do manuscrito para submissão.; 3. Certifica que o texto publicado está completamente isento de dados e/ou resultados fraudulentos; 4. Confirma a citação e a referência correta de todos os dados e de interpretações de dados de outras pesquisas; 5. Reconhece ter informado todas as fontes de financiamento recebidas para a consecução da pesquisa; 6. Autoriza a edição da obra, que incluem os registros de ficha catalográfica, ISBN, DOI e demais indexadores, projeto visual e criação de capa, diagramação de miolo, assim como lançamento e divulgação da mesma conforme critérios da Atena Editora.

DECLARAÇÃO DA EDITORA

A Atena Editora declara, para os devidos fins de direito, que: 1. A presente publicação constitui apenas transferência temporária dos direitos autorais, direito sobre a publicação, inclusive não constitui responsabilidade solidária na criação dos manuscritos publicados, nos termos previstos na Lei sobre direitos autorais (Lei 9610/98), no art. 184 do Código Penal e no art. 927 do Código Civil; 2. Autoriza e incentiva os autores a assinarem contratos com repositórios institucionais, com fins exclusivos de divulgação da obra, desde que com o devido reconhecimento de autoria e edição e sem qualquer finalidade comercial; 3. Todos os e-book são *open access, desta forma* não os comercializa em seu site, sites parceiros, plataformas de *e-commerce,* ou qualquer outro meio virtual ou físico, portanto, está isenta de repasses de direitos autorais aos autores; 4. Todos os membros do conselho editorial são doutores e vinculados a instituições de ensino superior públicas, conforme recomendação da CAPES para obtenção do Qualis livro; 5. Não cede, comercializa ou autoriza a utilização dos nomes e e-mails dos autores, bem como nenhum outro dado dos mesmos, para qualquer finalidade que não o escopo da divulgação desta obra.

Dissertação apresentada ao Curso de Pós-graduação em Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos para a obtenção de título de Mestre em Ciência da Computação. Orientador: Alexandre César Muniz de Oliveira

À minha mãe Joana; à minha esposa Marinilda; aos meus filhos Clerton Jr e Ludmila; e aos meus irmãos Sonia e Johny que me apoiaram plenamente ao longo de dois anos de caminhada.

AGRADECIMENTOS

Agradeço a Deus. Aos amigos que fiz no PPGCC nesses últimos 2 anos e meio, em especial Inêz, Bruno, Alan e Ricardo. A todos os professores do programa. Ao professor Alexandre César Muniz de Oliveira, meu orientador, que muito contribuiu no direcionamento da minha pesquisa. Por fim, meu agradecimento à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES pelo apoio financeiro a este trabalho.

SUMÁRIO

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CHESC	Cross-domain Heuristic Search Challenge
EPH	Evolutionary-Programming Hyper-heuristic
FUZZY LOGIC	Lógica difusa
GMP	Generalized Modus Ponens
IA	Inteligência artificial
ILS	Iterated Local Search
JVM	Java Virtual Machine
MBR	Master Boot Record
PCV	Problema do Caixeiro Viajante
PRV	Problema do roteamento de veículos
PRVC	Problema do roteamento de veículos capacitados
PRVCJT	Problema do roteamento de veículos capacitados com janelas de tempo
RAM	Random Access Memory
SO	Sistema Operacional
VND	Variable Neighborhood Descend

RESUMO

Existe, no mundo hoje, uma comunidade empenhada em descobrir novas formas de resolver problemas NP-hard. Tais problemas não admitem soluções em tempo de execução polinomial. Por isso, investem-se enormes esforços no estudo de alternativas que tornem possível a obtenção de soluções aproximadas através de algoritmos heurísticos. Neste cenário, surgem hiper-heurísticas, como um framework capaz de recomendar heurísticas de baixo nível específicas, bem como os ajustes de parâmetros necessários para um desempenho eficiente. A hiper-heurística usa um conjunto de heurísticas organizadas em um bloco separado do domínio do problema. Hiper-heurísticas competitivas, em geral, não estão totalmente emancipadas de ajustes de parâmetros, existindo quase sempre rudimentares mecanismos determinísticos de mudança de comportamento algorítmico. Esta dissertação propõe um mecanismo de controle adaptativo para equipar uma hiper-heurística aplicada ao problema de roteamento de veículos. Para tanto, foram realizados experimentos computacionais e análise dos resultados que dão evidências de notória melhora no desempenho para instâncias desafiadoras encontradas na literatura.

PALAVRAS-CHAVE: Otimização combinatória, Hiper-heurísticas, Heurísticas.

ABSTRACT

Nowadays, there is a community committed to discovering new ways of solving NP-hard problems. Such problems do not admit solutions in polynomial run time. Therefore, enormous efforts are invested in the study of alternative making it possible to obtain approximate solutions using heuristic algorithms. In this scenario, there are hyper- heuristics, as a framework able to recommend low-level specific heuristics as well as the parameter settings required for effective performance. The hyper-heuristic use a set of problem-specific heuristics, organized in a separate architectural block. Competitive Hyper-heuristics, in general, are not fully empowered of parameter settings, there often rudimentary deterministic mechanisms for algorithmic behavior change. This work proposes an adaptive control mechanism to equip a hyper-heuristic applied to vehicle routing problem. Therefore, computational experiments and analysis were conducted giving noticeable evidences of improvement of algorithm performance when applied to challenging instances found in the literature.
KEYWORDS: Combinatorial optimization, Hyper-heuristics, Heuristics.

CAPÍTULO 1 - INTRODUÇÃO

1 I CONTEXTO

1.1 Otimização

O contexto deste trabalho é a otimização combinatória, em que se busca a melhor solução candidata que atenda a uma série de requisitos e regras, a partir de um conjunto finito, mas quase sempre computacionalmente intratável.

A otimização combinatória é acima de tudo a arte de compreender um problema real, natural e ser capaz de transformá-lo em um modelo matemático. É a arte de enriquecer ou abstrair modelos existentes, para aumentar a sua força, portabilidade e capacidade para descrever matematica e computacionalmente outros problemas, que podem ou não ser semelhantes aos problemas que inspiraram os modelos iniciais. (PASCHOS, 2014).

1.2 Heurísticas

Segundo (HOAISS, 2009), uma heurística é um método de investigação baseado na aproximação progressiva de um dado problema. Para (FERREIRA, 2010), é uma metodologia, ou algoritmo, para resolver problemas por métodos que, embora não rigorosos, refletem o conhecimento humano e permitem obter uma solução satisfatória.

Heurísticas são procedimentos que tratam problemas de otimização sem dispor de garantias teóricas de que a solução ótima seja obtida, nem tampouco, de garantias de que a solução obtida tenha algum tipo de proximidade em relação à solução ótima exata procurada. A lógica de uma heurística é a de garantir que a solução obtida seja melhor que a vasta maioria das alternativas disponíveis, sendo nesse sentido, uma boa solução (GASPAR-CUNHA; TAKAHASHI; CARLOS HENGGELER, 2012).

Conforme define (PEARL, 1984a), heurística são critérios, métodos ou princípios para decidir qual, entre vários cursos alternativos de ação, seja o mais eficaz, com vistas a alcançar um determinado objetivo. Também é da natureza de uma boa heurística que ela proporcione um meio simples de ação, que mesmo não sendo o mais eficaz, forneça com frequência resultados suficientemente bons.

A maioria dos problemas complexos exigem a avaliação de um imenso número de possibilidades para determinar uma solução exata. O tempo necessário para encontrar um valor exato de uma solução é muitas vezes maior do que o tempo de uma vida. Por isso, heurísticas desempenham um papel eficaz na solução de tais problemas, indicando uma forma completa de reduzir o número de avaliações sob restrições de tempo razoável (PEARL, 1984a).

1.3 Meta-heurísticas

Os métodos de pesquisa meta-heurísticos podem ser definidos como metodologias que podem ser usadas para orientar estratégias na concepção de heurísticas subjacentes

na solução de problemas de otimização específicos (GLOVER; KOCHENBERGER, 2003). Para (LUKE, 2009), o termo meta-heurística é muito frequentemente usado para descrever uma grande subcampo, na verdade o subcampo primário da otimização estocástica. A otimização estocástica é uma classe geral de algoritmos e técnicas que empregam algum grau de aleatoriedade para encontrar as melhores soluções - ou o melhor possível - para problemas difíceis.

Meta-heurísticas são a classe mais genérica desses tipos de algoritmos e são aplicadas a uma ampla gama de problemas. São algoritmos usados para encontrar respostas para problemas quando não se sabe de antemão qual é a solução ideal; não se sabe como fazer para encontrar uma solução de maneira fundamentada; há pouca informação heurística para seguir em frente e por fim, a busca pela força bruta está fora de questão, porque o espaço é muito grande.

Para ilustrar o comportamento de uma meta-heurística, (LUKE, 2009) dá como exemplo a tentativa de encontrar um conjunto ideal de comportamentos para um robô goleiro de futebol: Um simulador para o robô pode testar qualquer conjunto de comportamento do robô e atribuir-lhe uma qualidade sabe-se quando um comportamento é bom quando o mesmo é visto. E se pode chegar a uma definição que avalia o comportamento do robô como sendo generalizado e bom. Mas não há percepção de que o conjunto de comportamento é ótimo, nem mesmo como fazer para encontrá-lo.

Métodos de otimização como meta-heurísticas são projetados para superar problemas diversos, são muito gerais, mas também significa que elas são muitas vezes melhor pensadas como única alternativa e usadas quando nenhuma outra técnica conhecida funciona (LUKE, 2009).

As meta-heurísticas representam uma família de técnicas de otimização aproximada que alcançaram muita popularidade nas últimas duas décadas, estando entre as mais promissoras e bem sucedidas técnicas de otimização (TALBI, 2009).

Ao contrário dos algoritmos de otimização exata, meta-heurísticas não garantem a otimização das soluções obtidas. Também, ao contrário dos algorítmos de aproximação, meta-heurísticas não definem o quão perto estão as soluções obtidas com base em um “ótimo”.

1.4 Hiper-heurísticas

Hiper-heurística, atualmente, é um termo muito citado na área de otimização combinatória. Não existe uma definição formal na língua portuguesa para definir hiperheurística, mas “hiper”, segundo definição de (FERREIRA, 2010), a palavra é originada do grego e significa “posição superior”.

A proposta básica de uma hiper-heurística é possibilitar a construção de um roteiro de processos que levem a solução de problemas computacionais tidos como intratáveis por meios exatos. Dessa forma, problemas de uma determinada classe são tratados não

com algoritmos específicos especializados, mas com várias soluções algorítmicas que somam suas vantagens e compensam suas desvantagens entre si para encontrar soluções razoáveis em um domínio de problema.

O resultado é obtido combinando-se e parametrizando-se tais heurísticas com o objetivo de se obter um resultado razoável em um conjunto generalizado de instâncias de problemas comuns. Basicamente, não é exigido que a hiper-heurística tenha conhecimento sobre o domínio do problema. Isso permite que ela atue sem que se tenha que modificar código.

A arquitetura de uma hiper-heurística é flexível e permite que se obtenha bons resultados sem a necessidade de analisar ou ajustar manualmente um algoritmo para um novo domínio de problema. Nesse ponto, as hiper-heurísticas têm certa similaridade com as meta-heurísticas, pois adotam uma abordagem caixa-preta para solucionar um problema.

Assim, hiper-heurísticas poderiam ser definidas como um tipo mais complexo de meta-heurística. Mas existe uma diferença que as torna únicas meta-heurísticas trabalham em um espaço de soluções-candidatas, já as hiper-heurísticas trabalham um espaço de busca de heurísticas simples denominadas heurísticas de baixo nível.

As hiper-heurísticas abrem caminho para uma nova ótica de abstração podendo ser consideradas como uma ampliação conceitual das meta-heurísticas. Isso vale tanto nos contextos da otimização com variáveis discretas quanto contínuas. O termo hiperheurística é relativamente novo, embora essa noção tenha sido sugerida em artigos de tempos em tempos, desde a década de 60 (ROSS, 2005).

Hiper-heurísticas podem apontar um conjunto de heurísticas que levem sistemas a lidar com uma grande variedade de domínios de problema, sem a especialização das meta-heurísticas, que tendem a focar um problema específico ou uma classe restrita de problemas. Hiper-heurísticas escolhem de forma inteligente o algoritmo correto ou o algoritmo heurístico que possa ser empregado sob determinadas situaçôes.

Grande esforço de pesquisa atual na ciência da computação consiste em desenvolver algoritmos que sejam rápidos e apresentem um bom desempenho sobre uma família inteira de problemas, presumindo-se que os algoritmos abordam algumas características comuns de todo o conjunto de tais problemas.

2 | OBJETIVO

Este trabalho tem como objetivo propor um dispositivo fuzzy autônomo de parametrização da profundidade de busca local para a hiper-heurística EPH (MEIGNAN, 2011). Essa hiper-heurística é formada por um conjunto de soluções construída a partir do framework HyFlex (Fig 1.1), que propõe cinco problemas de otimização, com diferentes heurísticas.

Mesmo com pouca parametrização, a EPH exige a intervenção do arquiteto de

software para fornecer empiricamente valores que atuam no controle da profundidade de busca local das heurísticas de baixo nível. Assim, a ideia de empregar um controlador fuzzy nessa tarefa, é reduzir o papel do especialista humano na parametrização da hiperheurística e consequente melhoria na qualidade das soluções obtidas para problemas de otimização combinatória do mundo real.

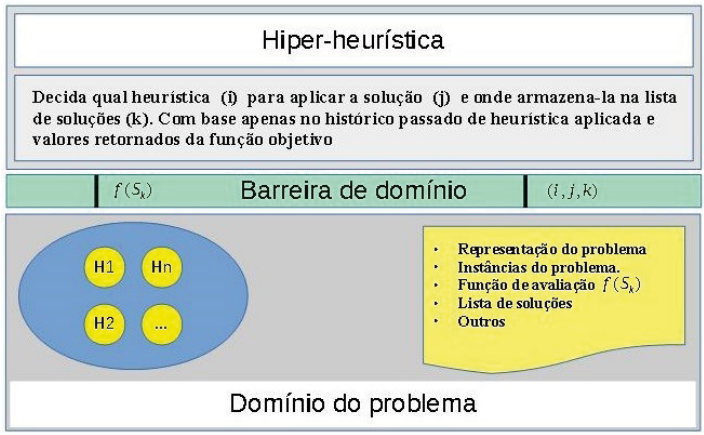


Figura 1.1: Arquitetura do framework HyFlex

A EPH foi implementada por (MEIGNAN, 2011) para o desafio CHeSC 2011[1], obtendo o quinto lugar no ranking.

A CHESC 2011 reuniu pesquisadores da área de pesquisa operacional, ciência da computação e inteligência artificial do mundo inteiro envolvidos no esforço de desenvolver uma metodologia a mais genérica possível aplicáveis na solução de problemas difíceis.

Basicamente, o desafio consistiu em projetar uma estratégia de pesquisa de alto nível que controlasse um conjunto de problemas específicos utilizando heurística de baixo nível. O conjunto de heurísticas de baixo nível foi diferente para cada domínio do problema, mas a estratégia de alto nível que controla a heurística permaneceu a mesma. A fim de executar as concorrentes estratégias de pesquisa de alto nível (hiperheurísticas) entre domínios diferentes, foi proposto o uso de uma interface de software comum para lidar com diferentes problemas de otimização combinatória. Nesse contexto, a HyFlex fornece um conjunto de rotinas para gerar e avaliar soluções.

O concurso foi organizado e executado pela Universidade de Nottingham[2] em parceria com a Universidade de Queen[3], Cardiff University[4] e École Polytechnique[5].

1 http:www.asap.cs.nott.ac.ukexternalchesc2011
2 http:http://www.nottingham.ac.uk/
3 http:www.qub.ac.uk
4 http:www.cardiff.ac.ukmaths
5 http:www.polymtl.ca

3 | ORGANIZAÇÃO DO TRABALHO

- O segundo capítulo deste trabalho trata do referencial teórico detalhado sobre heurísticas, meta-heurísticas, hiper-heurísticas e lógica fuzzy, iniciando com um pequeno histórico sobre otimização combinatória e sua evolução.
- O terceiro capítulo trata especificamente do problema de roteamento de veículos, um importante ramo da pesquisa na ciencia da computação, bem como trabalhos relacionados.
- O quarto capítulo apresenta a hiper-heurística EPH, construída a partir do framework HyFlex (Fig 1.1) e uma proposta de ajustes para melhorar a performance geral do conjunto de heurísticas de baixo nível.
- O capítulo cinco cuida especificamente dos experimentos conduzidos e os resultados obtidos.
- No último capítulo - seis - apresentam-se as conclusões e algumas ideias para trabalhos futuros.

CAPÍTULO 2 - REFERENCIAL TEÓRICO

1 | OTIMIZAÇÃO COMBINATÓRIA

Muitos problemas computacionais - teóricos ou práticos - do cotidiano consistem de se encontrar uma melhor configuração ou conjunto de parâmetros para alcançar algum objetivo. Ao longo das últimas décadas, uma hierarquia desses problemas tem surgido junto a um elenco correspondente de técnicas para sua solução. Na extremidade desta hierarquia existem os problemas de programação não-linear geral, do tipo Encontre *X* tal que:

$$Minimizar\ f(X)$$
$$Sujeito\ \ a\ \ g_i(X) \leq b_i \qquad para\ i = 1,2,...,m \tag{2.1}$$
$$X \geq 0 \tag{2.2}$$
$$com$$
$$X = (x_1,x_2,...,x_n) \tag{2.3}$$
$$f(.)\ e\ g_i(.) \quad \text{funções} \quad \text{não} \quad \text{lineares}$$

As técnicas para resolver tais problemas são quase sempre iterativas por natureza e sua convergência é estudada usando análise real matemática. A classe de problemas do que trata a Otimização combinatória fornece um conjunto finito de soluções candidatas. Por exemplo, o algoritmo simplex (DANTZIG, 1951), amplamente utilizado, encontra uma solução ótima na maioria das vezes para um problema de programação linear em número finito de passos.

Esse algoritmo (Fig 2.1) baseia-se na ideia de melhorar o custo ao passar de vértice para o vértice do politopo. Após trinta anos de refinamentos, as pesquisas levaram a várias versões do algoritmo simplex, que geralmente são consideradas como muito eficientes em termos de problemas com centenas de variáveis e milhares de restrições, resolvidos de forma rotineira.

SIMPLEX ALGORITHM

Input: A matrix $A \in \mathbb{R}^{m \times n}$ and column vectors $b \in \mathbb{R}^m, c \in \mathbb{R}^n$.
A vertex x of $P := \{x \in \mathbb{R}^n : Ax \leq b\}$.

Output: A vertex x of P attaining $\max\{cx : x \in P\}$ or a vector $w \in \mathbb{R}^n$ with $Aw \leq 0$ and $cw > 0$ (i.e. the LP is unbounded).

① Choose a set of n row indices J such that A_J is nonsingular and $A_J x = b_J$.

② Compute $c\,(A_J)^{-1}$ and add zeros in order to obtain a vector y with $c = yA$ such that all entries of y outside J are zero.
If $y \geq 0$ **then stop**. **Return** x and y.

③ Choose the minimum index i with $y_i < 0$.
Let w be the column of $-(A_J)^{-1}$ with index i, so $A_{J \setminus \{i\}} w = 0$ and $a_i w = -1$.
If $Aw \leq 0$ **then stop**.
Return w.

④ Let $\lambda := \min\left\{\dfrac{\beta_j - a_j x}{a_j w} : j \in \{1, \ldots, m\},\ a_j w > 0\right\}$,
and let j be the smallest row index attaining this minimum.

⑤ Set $J := (J \setminus \{i\}) \cup \{j\}$ and $x := x + \lambda w$.
Go to ②.

Figura 2.1: Algoritmo Simplex - Pseudocódigo extraído de BERNHARD; KORTE; VYGEN (2008)

No entanto, como afirma (WONG, 1983), existem problemas em que o algoritmo simplex pode ter de lidar com um número desagradavelmente exponencial de passos. Matemáticos soviéticos na década de 80, desenvolveram um algoritmo elipsóide (Fig 2.2) para programação linear que garante encontrar uma solução ideal em uma série de passos que cresce como um polinômio no tamanho do problema.

Basic Ellipsoid Algorithm

Input: Matrix A, vector b; sphere $S(c_0, R)$ containing P
Output: feasible $\overline{x}$ or proof that $P = \emptyset$

// Initialize
1: $k := 0$
2: $k^* = 2n(2n+1) < C > +2n^2(\log(R) - n^2 + 1)$ // Max number of iterations
3: $x^0 = c_0$, $B_0 = R^2 \cdot I$ // Initial ellipsoid
4: $\tau := \frac{1}{n+1}$ // Parameter for ellipsoid: step
5: $\sigma := \frac{2}{n+1}$ // Parameter for ellipsoid: dilation
6: $\delta := \frac{n^2}{n^2-1}$ // Parameter for ellipsoid: expansion
// Check if polytope P is empty
7: **if** $k = k^*$ **return** $P = \emptyset$
// Check feasibility
8: **if** $A\,x^k \leq b$ **return** $\overline{x} := x^k$ // $\overline{x}$ is a feasible point
9: **else** let $a_j^\top x^k > b_j$
// Construct new ellipsoid
10: $B_{k+1} := \delta(B_k - \sigma \frac{B_k a_j (B_k a_j)^\top}{a_j^\top B_k a_j})$
11: $x^{k+1} := x^k - \tau \frac{B_k a_j}{\sqrt{a_j^\top B_k a_j}}$
// Loop
12: $k \leftarrow k+1$ and goto step 7

Figura 2.2: Algoritmo Elipsoide - Pseudocódigo extraído de REBENNACK (2009)

Em certos programas lineares, problemas de fluxo e relacionados, podem ser resolvidos muito mais eficientemente do que aqueles solucionados por programas lineares gerais, como afirma (WONG, 1983). Por outro lado, tais problemas são também intimamente relacionados com a outra classe de problemas aparentemente intratáveis. Como exemplo, o problema de caminho mais curto em um grafo, que está nessa classe de problemas de fluxo e de correspondência, concluindo que de fato existe um algoritmo aproximado $O(n^2)$, para a sua solução, onde n é o número de nós no grafo.

Em contrapartida, existe o problema do caixeiro viajante, que pede o caminho mais curto fechado após visitar cada nó exatamente uma vez, que está na classe de problemas NP-completos, todos os quais amplamente considerados insolúveis por algoritmos polinomiais. Existe uma linha tênue entre problemas muito fáceis e problemas muito difíceis, isso é recorrente e tem atraído a atenção de milhares de designers de algoritmo ao redor do mundo.

A complexidade dos problemas NP-completos, as limitações dos dispositivos computacionais na década de 80 e o alto custo da implementação de soluções eficazes para tratar esse tipo de problema abriu espaço para o emprego de heurísticas.

2 | ALGORITMOS HEURÍSTICOS/APROXIMADOS

Heuriskein (em grego antigo) e heurisricus (Latin), significa “para descobrir,

descobrir."(HOAISS, 2009) define heurística como sendo arte de inventar, de fazer descobertas; ciência que tem por objeto a descoberta dos fatos, apresentando uma rubrica específica para informática como sendo um método de investigação baseado na aproximação progressiva de um dado problema

Desde a metade dos anos 50 até o presente momento, a definição formal do que seria uma heurística tem desempenhado papel crucial na definição de cada pesquisador que esteve envolvido no trabalho com otimização. Coisas que são definidas como uma heurística por um pesquisador não são reconhecidas por outros como tal. Isso se dá porque muitas heurísticas incorporam uma variedade de características diferentes, o que leva pesquisadores (PEARL, 1984b) (TVERSKY; KAHNEMAN, 1974) (MICHALEWICZ; FOGEL, 2013) a enfatizar diferentes desses recursos como sendo essencial para que uma solução seja considerada uma heurística.

Um trabalho pioneiro de (ROMANYCIA; PELLETIER, 1985) procura conceituar e definir claramente o papel de uma heurística. O autor lembra que o conceito de heurística tem sido sempre, o foco central da IA, mas que apesar de muito bem conhecido, exige melhor documentação. Afirma ainda que ao longo da história da sua utilização em IA, diferentes teóricos têm empregado diferentes definições pessoais para uma heurística, de modo que o que já foi definido como sendo uma clara instância de uma heurística mais tarde seria visto apenas como uma instância marginal. Ele apresenta quatro dimensões para o estudo de heurísticas:

- Incerteza do resultado
- Conhecimento incompleto
- Melhoria de desempenho
- Orientação de tomada de decisão

Como o objetivo do presente trabalho não é um estudo sobre heurísticas, a conceituação definida por (ROMANYCIA; PELLETIER, 1985) é considerada.

Dessa forma, heurística pode referir-se a qualquer dispositivo utilizado em resolução de problemas, seja ele um programa, uma estrutura de dados, um provérbio, uma estratégia ou uma fração de conhecimento. Entretanto, deve haver um elemento de regra sobre o dispositivo, que tem de ser útil, mas não imprescindível para garantir o sucesso.

Segundo ainda (ROMANYCIA; PELLETIER, 1985), um dispositivo heurístico pode garantir o fornecimento de uma solução, mas não pode garantir que ele também seja comprovadamente a melhor forma de se chegar a uma solução. O autor segue estabelecendo um padrão desejável para uma heurística como sendo algo que apresente melhoria significativa de desempenho e qualidade da solução.

De forma sucinta, (ROMANYCIA; PELLETIER, 1985) define uma heurística como sendo qualquer dispositivo, seja ele um programa, regra ou conhecimento, que não é

inteiramente confiável, mas que será útil para proporcionar uma solução prática e que uma vez adicionada a um sistema de resolução de problemas, vai colaborar para melhorar o resultado.

O conceito de heurística depende do conceito de algoritmo, mas não tem o mesmo significado. A diferença entre um algoritmo e uma heurística é bastante sutil. Sob alguns aspectos, os dois termos se sobrepõem.

A principal diferença entre os dois é o nível de acurácia obtido na solução. Um algoritmo recebe instruções diretamente e as executa. Uma heurística aponta um rumo a seguir.

Um algoritmo é um conjunto de instruções bem definidas para a realização de uma determinada tarefa. Essa tarefa deve ser consistente e completa. Isso significa que ele sempre dá uma resposta correta e deve funcionar para todos os casos. Usualmente, um algoritmo é previsível, determinístico, e não está sujeito ao acaso.

Uma heurística, entretanto, é uma técnica que ajuda a procurar uma resposta. Os seus resultados estão sujeitos a valores pobres, já que a heurística diz como olhar, não o que encontrar. Uma heurística não diz como chegar diretamente de um ponto A a um ponto B. Ela pode até não saber onde está o ponto A e o ponto B. Assim, uma heurística é um algoritmo sem o determinismo e a formalidade da previsão.

Durante muito tempo, as heurísticas foram consideradas como a melhor forma de se resolver problemas difíceis, de resultados incertos. Elas são constituidas de regras baseadas na experiência e no planejamento e substituem ou complementam regras baseadas na procura algorítmica que chega a soluções corretas depois de ter combinado o problema com todas as soluções permitidas.

Métodos heurísticos procuram um grau tão alto quanto possível de uma ação perante uma situação. Essa metodologia soma estratégias, procedimentos, métodos de aproximação por tentativa e/ou erro, buscando sempre a melhor forma de chegar a determinado objetivo.

Um procedimento heurístico pode exigir na maioria das vezes menos tempo que um processo algorítmico e se aproxima mais da forma como o ser humano pensa e chega a uma solução de um problema, podendo oferecer soluções eficientes.

Uma busca heurística é uma busca realizada pela quantificação do quão próximo um resultado está do objetivo. Pode-se afirmar que uma heurística é boa, se o objeto avaliado está muito próximo do objetivo final a ser alcançado. Assim, algoritmos de aproximação podem ser classificados como heurísticos, pois utilizam conhecimento e predição sobre a instância do problema e da sua estrutura para encontrar uma solução rápida. Em contraponto, nem todo algoritmo heurístico pode ser definido como aproximativo. Uma heurística não tem previamente sua qualidade comprovada matematicamente ou uma prova formal que terá sempre resultados satisfatórios. Por isso, alguns pesquisadores (PEARL, 1984b) (TVERSKY; KAHNEMAN, 1974) (MICHALEWICZ;

FOGEL, 2013) separam claramente esses termos, como segue:

- Algoritmos aproximativos - Entregam soluções dentro de um limite de qualidade efetiva/assintótico, bem como um limite assintótico polinomial de complexidade para pior caso comprovado matematicamente;
- Heurísticas - Entregam soluções sem limite formal de qualidade, empiricamente, em termos de complexidade média e qualidade das soluções. Trata-se de um conjunto de regras e métodos que conduzem à descoberta, invenção e solução de problemas sobre os quais se tem pouca ou nenhuma forma de resolver.

O emprego de heurísticas é justificado quando:

- Não se pode compreender - de imediato - um problema
- Não se pode encontrar a solução, diretamente - recorre-se à engenharia reversa para se chegar à solução
- Estuda-se um problema abstrato - tenta-se propor o mesmo problema em exemplo concreto
- Existem muitos detalhes - avança-se do nível mais genérico para convergir no sentido da especialização

Um exemplo clássico do emprego de heurística é a detecção de vírus de computador. O quadro atual é bastante diferente de meados dos anos 80, quando se massificou o uso de microcomputadores. Para combater a ação de vírus na sua apresentação atual, muito esforço foi realizado no sentido de se criar tecnologias inovadoras e eficazes na detecção de vírus.

Entre 1980 e 1990, os softwares antivírus utilizavam técnicas bastante rudimentares de busca, em um cenário bem pobre. Um típico disco rígido dos anos 80 se limitava a 32 MB máximo de capacidade. A pesquisa nesse espaço era rápida e o tipo de objeto procurado estava restrito a famílias específicas de arquivos do sistema - tipicamente arquivos .COM e .EXE e o MBR do SO.

Discos cresceram de forma quase exponencial, passando a suportar gigabytes e terabytes, atualmente. Além disso, muitas famílias de arquivos passaram para a lista de infectáveis. As soluções tiveram de ser trabalhadas de forma radical, pois se necessitava melhorar a resposta na identificação e remoção de infecção em muitos formatos distintos.

Hoje, grande parte dos vírus ainda pode ser detectada pela sua assinatura binária na forma de pequenas porções de dados utilizados para identificar um determinado padrão virótico.

A heurística é utilizada para estudar o comportamento, a estrutura e as características de um arquivo, definindo-o como suspeito ou não. Considerando-se toda a complexidade envolvida no processo de identificar ameaça, um antivírus moderno consegue ser eficaz

entre 70% e 90% das vezes em que é instado a apontar uma ameaça desconhecida.

3 | META-HEURÍSTICAS

Meta-heurística é um termo bastante usado para descrever uma grande subcampo da otimização estocástica - classe geral de algoritmos e técnicas que empregam algum grau de aleatoriedade para encontrar uma solução razoável ou próxima disso em um cenário de possíveis soluções para problemas difíceis. As meta-heurísticas são os mais genéricos desses tipos de algoritmos e são aplicados a uma ampla gama de problemas.

Conforme definição de (LUKE, 2009), meta-heurísticas (Fig 2.3) são ferramentas utilizadas para encontrar respostas para os problemas quando se tem muito pouco para iniciar. Não se sabe de antemão que solução ideal parece boa, não se sabe como fazer para encontrar tal solução, tem-se pouca informação heurística para continuar e a pesquisa pela força bruta está fora de questão porque o espaço é muito grande. Mas se existe uma solução candidata para o problema, é possível testá-la e avaliar o quão boa ela é, ou seja, será possível ver que é uma boa solução quando a mesma for vista.

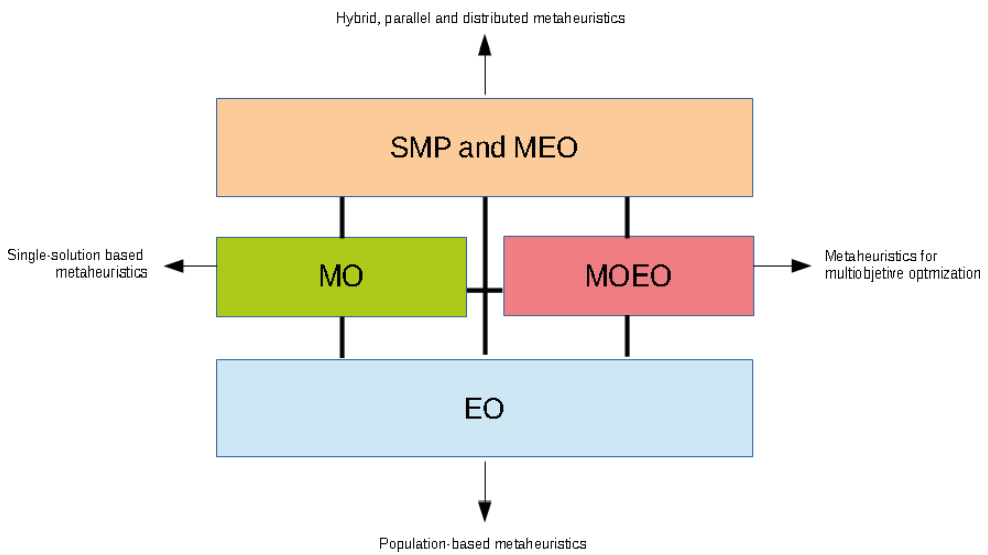


Figura 2.3: Arquitetura de uma meta-heurística - o Framework Paradiseo[1]

O algoritmo hill-climbing (Fig 2.4) é um exemplar ímpar dessa classe. Trata-se de uma técnica simples de otimização matemática que pertence à família de algoritmos baseados em busca local. É iterativo, inicia com uma solução arbitrária para um problema dado, tenta obter uma melhor solução, para isso muda incrementalmente um único elemento da solução. Se a mudança produzir uma solução melhor, uma mudança incremental é feita

1 http:paradiseo.gforge.inria.fr

para a nova solução, repetindo até que não seja mais possível se obter melhorias.

Stochastic Hill Climbing.

```
Input: Iter_max, ProblemSize
Output: Current
Current ← RandomSolution(ProblemSize);
foreach iter_i ∈ Iter_max do
    Candidate ← RandomNeighbor(Current);
    if Cost(Candidate) ≥ Cost(Current) then
        Current ← Candidate;
    end
end
return Current;
```

Figura 2.4: Algoritmo Hill Climbing - Pseudocódigo extraído de BROWNLEE (2011)

Esse algoritmo sempre vai encontrar fácil uma solução inicial para o PRVC ou para o PCV, por exemplo. Entretanto, essa solução será muito pobre se comparada com a solução ótima. O algoritmo é iniciado com uma solução qualquer e na sequencia realiza pequenas melhorias tais como alternar a ordem de visita a um par de vértices. A relativa simplicidade do algoritmo faz com que seja uma primeira escolha popular entre os algoritmos de otimização.

Embora existam algoritmos mais avançados, tais como recozimento simulado ou busca tabu e estes possam fornecer melhores resultados, em algumas situações hillclimbing pode produzir resultados melhores do que esses algoritmos quando a quantidade de tempo disponível para executar uma pesquisa é limitada, tal como com sistemas de tempo real. Ele pode até mesmo retornar uma solução válida mesmo que seja interrompido a qualquer momento antes de terminar.

Essa crença heurística, segundo ainda (LUKE, 2009), é uma das características do conceito central das meta-heurísticas, a de que, na verdade, quase todas as metaheurísticas são essencialmente combinações elaboradas de hill-climbing e busca aleatória (Fig 2.5).

Adaptive Random Search.

Input: $Iter_{max}$, $Problem_{size}$, SearchSpace, $StepSize^{init}_{factor}$, $StepSize^{small}_{factor}$, $StepSize^{large}_{factor}$, $StepSize^{iter}_{factor}$, $NoChange_{max}$
Output: S

$NoChange_{count} \leftarrow 0$;
$StepSize_i \leftarrow$ `InitializeStepSize(SearchSpace,` $StepSize^{init}_{factor}$`)`;
$S \leftarrow$ `RandomSolution(`$Problem_{size}$`, SearchSpace)`;
for $i = 0$ **to** $Iter_{max}$ **do**
  $S_1 \leftarrow$ `TakeStep(SearchSpace,` S`,` $StepSize_i$`)`;
  $StepSize^{large}_i \leftarrow 0$;
  if $i \bmod StepSize^{iter}_{factor}$ **then**
    $StepSize^{large}_i \leftarrow StepSize_i \times StepSize^{large}_{factor}$;
  else
    $StepSize^{large}_i \leftarrow StepSize_i \times StepSize^{small}_{factor}$;
  end
  $S_2 \leftarrow$ `TakeStep(SearchSpace,` S`,` $StepSize^{large}_i$`)`;
  if `Cost(`S_1`)`$\leq$`Cost(`S`)` $--$ `Cost(`S_2`)`$\leq$`Cost(`S`)` **then**
    if `Cost(`S_2`)`$<$`Cost(`S_1`)` **then**
      $S \leftarrow S_2$;
      $StepSize_i \leftarrow StepSize^{large}_i$;
    else
      $S \leftarrow S_1$;
    end
    $NoChange_{count} \leftarrow 0$;
  else
    $NoChange_{count} \leftarrow NoChange_{count} + 1$;
    if $NoChange_{count} > NoChange_{max}$ **then**
      $NoChange_{count} \leftarrow 0$;
      $StepSize_i \leftarrow \frac{StepSize_i}{StepSize^{small}_{factor}}$;
    end
  end
end
return S;

Figura 2.5: Busca aleatória - Pseudocódigo extraído de BROWNLEE (2011)

Segundo (LUKE, 2009), problemas abordados por meta-heurísticas são uma subclasse de problemas inversos. Um problema inverso é aquele em que se você tem uma função de teste f que leva uma solução candidata e produz uma avaliação da mesma, mas em que é difícil ou impossível de construir a função inversa f^{-1}, que leva a uma avaliação e retorna uma solução que seria uma solução candidata.

Métodos de otimização como as meta-heurísticas são projetados para superar os problemas inversos. Meta-heurísticas são muito genéricas, mas também são muitas vezes a melhor ideia que se pode ter como última alternativa, quando nenhuma outra técnica conhecida funciona.

Apesar das peculiaridades, meta-heurísticas possuem três características comuns - possibilidade de soluções desconexas, uma função objetivo e, por último, um critério de

parada.

- Basicamente, o critério de parada é uma forma de lidar com as soluções desconexas ou com a estagnação - tarefas que uma meta-heurística não controla. Ao projetista de software cabe a tarefa de resolver isso. Mais trivial é a opção por relaxação do problema, tal que todas as soluções se tornam viáveis, fazendo com que a função objetivo aplique penalidade contra soluções desconexas. Mas também é possível empregar um arremedo de heurística, como por exemplo, impedindo o tratamento de soluções não viáveis.
- Quando existe a necessidade de se avaliar uma solução que exija alto custo computacional, é importante se verificar a função objetivo. Quando isso acontece, indica-se a aproximação simplificada da função objetivo. Uma outra alternativa é a implementação de uma regra para avaliar toda nova solução a partir de outras soluções que tenham dado origem a ela.
- O instante em que um algoritmo deve ter encerrada a sua execução chama-se critério de parada. Pode-se definir esse critério levando em conta fatores específicos, não contemplados pela inteligência da meta-heurística. Dentre tantos fatores possíveis, pode-se citar a quantidade de iterações realizadas, quantidade de iterações seguintes que não melhoraram a melhor solução e tempo da execução da instância avaliada.
- Na classificação de meta-heurísticas, é possível se considerar aspectos que podem ser bem relevantes e determinísticos. Porém, caso se leve em conta características mais triviais, pode-se observar que existem apenas duas classes relevantes e distintas, as quais englobam praticamente tudo o que pode ser chamado de meta-heurísticas. Assim, as metodologias meta-heurísticas podem ser classificadas como de busca em vizinhança e populacionais.

4 | HIPER-HEURÍSTICAS

As hiper-heurísticas surgiram com o objetivo de diferenciar os algoritmos propostos existentes das demais estratégias de otimização. A ideia é não se usar informações específicas do domínio do problema e sim heurísticas simples de baixo custo de implementação/manutenção mais uma função que avalie o custo de uma determinada solução.

Dessa forma, ao desenho de novos algoritmos de otimização combinam-se heurísticas de tal sorte que uma compense as deficiências das outras (BURKE et al., 2003a). O conceito de hiper-heurística está atrelado a uma evolução histórica de metodologias e práticas e está intimamente conectado ao aperfeiçoamento/barateamento do hardware computacional.

É fato que algoritmos que se baseiam em meta-heurísticas apresentam resultados sólidos em muitos domínios de problemas. Mas a sua implementação, na prática, está

vinculada a apenas um domínio de problema por vez. Não existe a possibilidade de se ter uma solução suficientemente generalizada que contemple toda a sorte de problema, mesmo sob mesmo domínio.

Mesmo as meta-heurísticas estando entre as mais promissoras técnicas de otimização de hoje, podem fornecer soluções apenas “aceitáveis”em um tempo razoável para a resolução de problemas difíceis e complexos da ciência e da engenharia. Ao contrário dos algoritmos de otimização exata, metaheurísticas não garantem a otimização das soluções obtidas (TALBI, 2009).

Entre a variedade de métodos de busca existente destacam-se os métodos de pesquisa local: eles iniciam a partir de alguns pontos escolhidos. Efetuam melhorias através de pesquisa em algum tipo de vizinhança. Quando uma melhoria é encontrada, o processo é reiniciado a partir dessa posição melhorada. Muitas vezes, a ordem na qual as melhorias candidatas são consideradas é determinada por algum processo de seleção heurística.

As muitas categorias de métodos de busca local incluem meta-heurísticas, tais como busca na vizinhança variável que ajusta o tamanho da vizinhança, e busca local guiada (TSANG; VOUDOURIS, 1997), que ajusta o valor ou fitness de uma solução conforme a busca progride.

A pesquisa atual sobre a hiper-heurísticas tenta resolver velhos problemas de precisão dos métodos heurísticos. O objetivo geral é encontrar algum algoritmo que resolva uma gama inteira de problemas, que seja rápido, razoavelmente compreensível, confiável em termos de qualidade e repetibilidade e com bom comportamento de pior caso em toda essa gama de problemas (BURKE et al., 2003b). O espaço de possíveis algoritmos para qualquer tipo de problema, é grande.

Um dos primeiros exemplos do que poderia ser chamado de uma abordagem hiper- heurística foi a solução utilizada dentro do sistema COMPOSER (GRATCH; CHIEN; DEJONG, 1993). Essa solução se propunha a planejar horários de comunicação entre a órbita da Terra, satélites e estações terrestres, com um intervalo máximo entre comunicação com um dado satélite. As estações terrestres que poderiam ser utilizadas eram restringidas por órbitas do satelite.

O programador utilizava métodos heurísticos para tentar construir uma agenda, decidir qual restrição era insatisfeita e resolver como tentar satisfazê-las. Várias heurísticas foram disponibilizadas para se usar em cada caso, utilizando uma abordagem hill-climbing simples para investigar as combinações delas, testando cada conjunto em 50 problemas diferentes, o algoritmo foi capaz de descobrir uma combinação eficaz.

Seguindo a corrente de bons resultados, (TERASHIMA-MARÍN; ROSS, 1999) aplicaram hiper-heurísticas para resolver uma série de problemas de calendário de exames universitários.

Como afirma (BURKE et al., 2003b), quando se estuda o emprego de hiper-heurísticas, surge sempre a mesma pergunta: como deve ser o processo de escolha de

uma Heurística? Pergunta-se também quão sensíveis os resultados podem ser no processo e o melhor caminho a seguir é, provavelmente, a realização de experiências sistemáticas, claramente, no interesse de gerar um algoritmo que termina; cada heurística utilizada deve fazer algum progresso discernível em direção a uma solução; se não, o algoritmo pode escolhê-la repetidamente e portanto entrar em loop contínuo.

Além disso, (BURKE et al., 2003b) afirma que embora alguns autores empregaram em um determinado momento o conceito de heurística aleatória (escolha aleatória de uma heurística durante o processo de resolução do problema), seria sensato excluir o uso de tais heurísticas aleatórias do algoritmo final, porque o algoritmo estaria aberto às mesmas críticas de não-repetição e incompreensibilidade.

Entretanto, existe um lugar para o uso de heurísticas aleatórias durante a fase de desenvolvimento: se a inclusão aleatória de uma heurística melhora os resultados em seguida, ele sugere que outras heurísticas entre elas não são tão capazes de gerar bom desempenho e, portanto, esse conjunto de heurísticas talvez precise ser prorrogado ou alterado (BURKE et al., 2003b). O autor prossegue recomendando cautela ao se inserir um conjunto muito grande de heurísticas, realizando assim, uma espécie de ataque maciço ao problema escolhido. Deve-se ter em mente também que, às vezes, um indivíduo é excelente mas não adequado para ser usado em uma combinação hiper- heurística. Cita o caso do algoritmo guloso Kruskal (KRUSKAL, 1956) para gerar árvore geradora mínima em grafo ponderado sempre encontra soluções ótimas e que portanto talvez não possa ser considerado como uma heurística.

Adiante, (BURKE et al., 2003b) questiona se um algoritmo de busca deve ser usado, considerando que ainda não está claro que tipo de método de pesquisa deve ser empregado. Depende muito da natureza do espaço de busca e não simplesmente do seu tamanho. Sugere ainda que algoritmos evolutivos são uma boa maneira de pesquisar espaços enormes. Lembra ainda que em qualquer situação dada, ao considerar uma heurística para aplicar, pode muito bem ser o caso em que várias heurísticas poderiam gerar a mesma escolha. Assim, pode haver muitas soluções, mesmo em um espaço grande e métodos de busca local, tais como hill-climbing podem, às vezes, ser a melhor escolha em termos de custo para encontrar um algoritmo aceitável.

Sobre a avaliação de desempenho durante a busca, (BURKE et al., 2003b) considera que alguns autores têm utilizado métodos de hiper-heurística para resolver problemas individuais e para eles, a questão de como avaliar um algoritmo gerado pela hiper- heurística é fácil: deve-se simplesmente aplicá-lo ao problema. Por outro lado, se o que se está tentando encontrar é um algoritmo que apresente bom desempenho através de um grande conjunto de problemas, isso fica difícil de determinar.

Ainda nesse contexto, (BURKE et al., 2003b) recomenda que isso não seja feito, indicando uma alternativa simples proposta pela primeira vez em (GATHERCOLE; ROSS et al., 1997): a escolha de alguns problemas inicialmente para fins de avaliação, mas também

manter o controle de quando cada problema foi usado pela última vez para a avaliação e como ele foi resolvido quando utilizado.

É recomendado ainda que a cada ciclo subsequente, escolha-se um pequeno subconjunto de avaliação de uma forma que leve em conta tanto suas aparentes dificuldades e tempo decorrido do último caso de uso. O tamanho do grupo escolhido pode também crescer com o tempo, de modo que como a busca estreita em direção a um bom algoritmo, os candidatos são testados em um subconjunto crescente (BURKE et al., 2003b).

Sobre que tipo de algoritmo deve ser procurado, (BURKE et al., 2003b) pondera que para muitos problemas, os melhores métodos de pesquisa existentes para resolver problemas individuais começam pela construção de uma solução inicial completa ou talvez uma coleção delas e em seguida, deva-se procurar por melhorias com métodos de busca local guiados heuristicamente. A investigação hiper-heurística talvez tenda a se concentrar em encontrar soluções de construção incrementais, mas aquelas que modificam soluções candidatas completas podem despontar como a melhor alternativa.

5 | LÓGICA FUZZY

A Lógica Fuzzy - ou Lógica nebulosa - foi proposta por (ZADEH, 1965) e contempla um conjunto de métodos bastante fundamentados para modelagem e raciocínio sobre conceitos imprecisos ou aproximados.

Conceitos e prática foram expandidos por (MAMDANI, 1974), levando essa área da IA a contabilizar milhares de publicações relacionadas ao desenvolvimento teórico realizado neste campo e às muitas aplicações práticas implementadas em áreas específicas tais como sistemas de apoio à decisão, agrupamentos de dados, classificação de padrões, sistemas de diagnósticos, processamento de imagens e sistemas de controle.

A Teoria dos Conjuntos Nebulosos - ou difusos - pode ser classificada como uma extensão da teoria dos conjuntos e a forma como as informações são representadas e processadas neste contexto pode ser vista como uma extensão da lógica clássica, ficando conhecida como Lógica Fuzzy (LIMA; PINHEIRO; SANTOS OLIVEIRA, 2004).

Na lógica tradicional, consideram-se dois valores para determinar a verdade sobre uma afirmação - verdadeiro ou falso e a sua representação correspondente na álgebra booleana como sendo “1”ou “0”, respectivamente. Dessa forma, problemas só admitem como limites, por exemplo: a “água está quente ou fria?”. Nesse exemplo, a água até poderia estar nem tão quente, não tão fria, ou quaisquer outros enquadramentos que significassem não estar muito quente. A Lógica Fuzzy (Fig 2.6) lida com as fronteiras de um conceito, podendo as coisas serem “um pouco verdade”e “longe da verdade”.

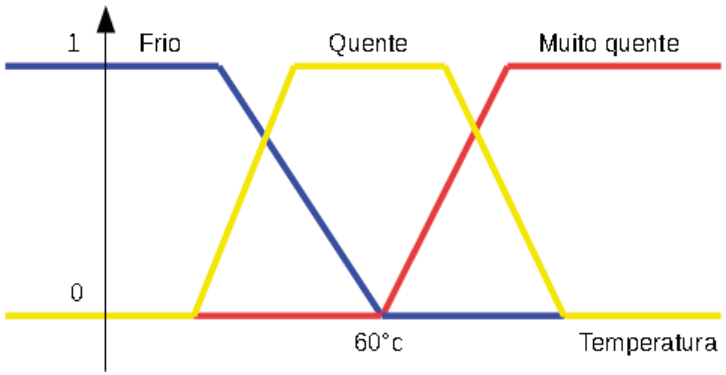


Figura 2.6: Lógica Fuzzy para obter uma resposta não booleana

Em um conjunto nebuloso o grau de pertinência associado a um determinado conjunto também possui os valores limites “0” e “1”, mas pode assumir infinitos valores na faixa [0, 1].

Um conjunto fuzzy *F* em um universo de discurso *U* , caracteriza-se por uma função de pertinência de grau μ_f que assume valores no intervalo [0, 1], ou seja, $\mu_f : U \rightarrow [0,1]$. Genericamente, um conjunto fuzzy *F* em *U* pode ser representado como um conjunto de pares ordenados compostos de valores de uma variável e os respectivos valores de grau de pertinência μ_f (u), com a notação expressa pela equação (2.1). (LIMA; PINHEIRO; SANTOS OLIVEIRA, 2004)

$$F = (u,\mu_F(u))|u \in U \qquad (2.1)$$

A Figura 2.7 representa os parâmetros de uma função de pertinência triangular típica.

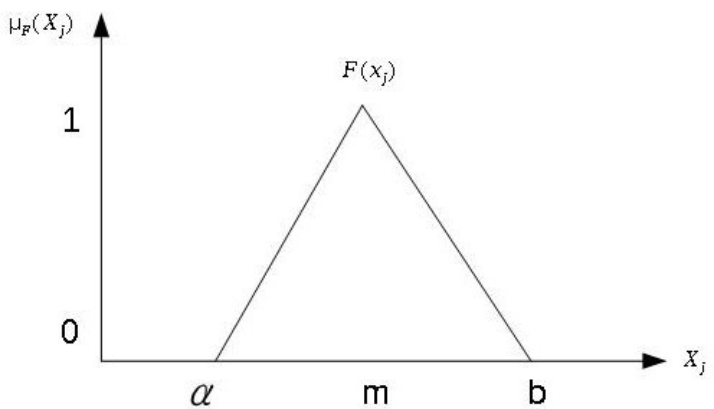


Figura 2.7: Função de pertinência triangular - Retirado de (LIMA; PINHEIRO; SANTOS OLIVEIRA, 2004)

Na Figura 2.8, o grau de pertinência relativo a cada conjunto nebuloso associado ao tamanho e os seus respectivos estados (Pequeno, Médio e Alto) é simbolizado por $\mu_F(x_1)$. Por exemplo, para $x_1 = 2$ têm-se: *Pequeno*$(x_1) = 0$; *Medio*$(x_1) = 1$; *Alto*$(x_1) = 0$.

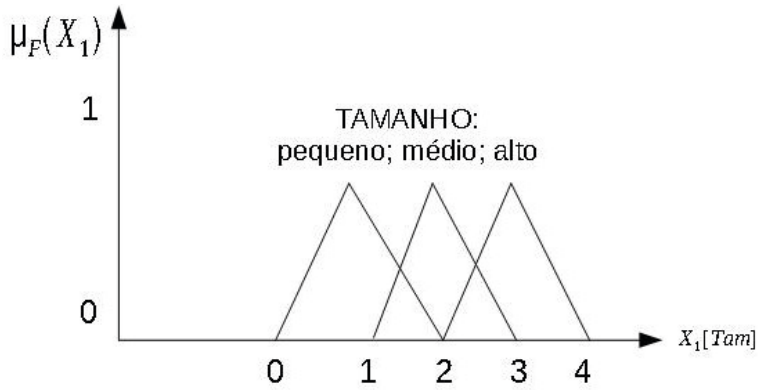


Figura 2.8: Ilustração gráfica de conjuntos nebulosos - Retirado de (LIMA; PINHEIRO; SANTOS OLIVEIRA, 2004)

5.1 Fuzificação

A conversão da informação de tamanho ou qualquer outra grandeza em um grau de pertinência entre [0, 1] é conhecida como fuzificação.

Na etapa de fuzificação são definidas variáveis linguísticas, de forma subjetiva, bem como as funções membro/pertinência.

Essa etapa determina

1. Análise do Problema
2. Definição das Variáveis
3. Definição das Funções de Pertinência. Para cada variável, diversos tipos de espaço podem ser gerados. Os espaços mais comuns são: Triangular, Trapezoidal, Singleton e Shouldered
4. Criação das Regiões

É possivel se representar informações linguísticas no formato de rótulos tais como Pequeno, Médio, Alto, Baixo. Isso simplifica a modelagem para uma forma qualitativa e não apenas quantitativa.

5.2 Variáveis linguísticas

É a base da técnica de modelagem de sistemas fuzzy. Uma variável linguística é o nome do conjunto fuzzy e pode ser usado num sistema baseado em regras para tomadas de decisão

Exemplo: if objeto.valor is 4 then tamanho is GRANDE.

Isso permite que a linguagem da modelagem fuzzy expresse a semântica usada por especialistas. Encapsula as propriedades dos conceitos imprecisos numa forma usada computacionalmente

5.3 Inferência

Nessa etapa as proposições ou regras são definidas e depois são examinadas paralelamente e envolve:

- Definição das proposições
- Análise das Regras
- Criação da região resultante

5.4 Defuzificação

Na defuzificação, as regiões resultantes são convertidas em valores para a variável de saída do sistema. Essa etapa corresponde a ligação funcional entre as regiões Fuzzy e o valor esperado.

Dentre os diversos tipos de técnicas de defuzificação destacam-se: Centróide, First-of-Maxima, Middle-of-Maxima e Critério Máximo.

5.5 Conjuntos nebulosos

São funções que mapeam o valor que poderia ser um membro do conjunto para um número entre 0 e 1. O grau de pertinência 0 indica que o valor não pertence ao conjunto O grau 1 indica significa que o valor é uma representação completa do conjunto. Um conjunto fuzzy indica com qual grau um valor específico é membro do conjunto de tamanho GRANDE. A definição do que é GRANDE depende do contexto

5.6 Operações sobre conjuntos nebulosos

As operações básicas sobre conjuntos nebulosos são definidas a partir de normas definidas da teoria dos conjuntos, como as descritas a seguir.

T-norma

Operação binária que satisfaz as condições de:

- Comutatividade: x t y = y t x;
- Associatividade: x t (y t z) = (x t y) t z;
- Monotonicidade: $x \leq y$ e $w \leq$ z então x t $w \leq y$ t z;
- Condições limite: 0 t x = 0; 1 t x = x

S-norma

Operação binária que satisfaz as condições de:

- Comutatividade: x s y = y s x;
- Associatividade: x s (*ys z*) = (*x s y*) s z;
- Monotonicidade: $x \leq y$ e $w \leq z$ então $x\ s\ w \leq y\ s\ z$;
- Condições limite: 0 *s x* = x; 1 s x = 1

Na álgebra booleana as operações típicas compreendem as seguintes funções lógicas básicas: “AND”; “OR”; “NOT”. Em conjuntos nebulosos foram definidas operações típicas baseadas em t-normas e s-normas, onde as operações básicas são o mínimo (min), o produto algébrico (*) e o máximo (max). O conceito do complemento de um conjunto nebuloso *A*(x) é definido como *C*(x) = 1 – *A*(x).

A Lógica Fuzzy reduz a complexidade do sistema, evitando intrincadas formulações matemáticas e lógicas necessárias para resolver problemas nessas condições de uso do existe / não existe. Ela é capaz de aproximar o comportamento de problemas que são mal compreendidos. A Lógica Fuzzy também apresenta outra característica: permite que o usuário represente ou modele o problema em linguagem humana, levando à compreensão do problema de forma natural para o usuário.

Na pergunta “água está quente ou fria?”é possível o emprego de variáveis linguísticas (semelhantes às variáveis regulares como X e Y, podendo-se classificar valores como “frio”ou “não muito frio”, obtendo-se assim um mapeamento mais estreito entre o mundo real, o problema e a representação computacional do problema. Ao todo, a Lógica Fuzzy fornece um mecanismo para reduzir o tempo de modelagem e desenvolvimento de sistemas não-lineares complexos, aumentando assim a compreensão e entendimento das soluções obtidas.

A Teoria dos conjuntos tradicional exige que elementos podem ser parte de um conjunto ou não, como afirma (ENGELBRECHT, 2007). Da mesma forma, a lógica de valor de binário limita os valores de parâmetros a 0 ou 1, com bastante restrições sobre o resultado de um processo de inferência. O raciocínio humano é, no entanto, quase sempre não exato. Segundo o autor, observações e raciocínio costumam considerar uma medida de incerteza. Adiante, (ENGELBRECHT, 2007) enfatiza que os seres humanos são capazes de entender a frase: “Alguns estudantes de Ciência da Computação podem programar na maioria das linguagens”.

Tal afirmação, entretanto, não pode ser representada, nem deduzida por um computador. Nesse contexto, entra a teoria dos Conjuntos fuzzy e a Lógica fuzzy, permitindo se chegue no que é conhecido como o raciocínio aproximado.

Com os conjuntos fuzzy, um elemento pertence a um conjunto a partir de um certo grau de certeza. A Lógica fuzzy permite raciocinar como esses fatos incertos podem inferir

fatos novos, com um grau de certeza associado a cada fato.

A incerteza em sistemas fuzzy é referido como não incerteza estatística e não deve ser confundido com a incerteza estatística. Incerteza estatística é baseada na leis da probabilidade, enquanto a incerteza não estatística é baseada em imprecisão, imprecisão e / ou ambiguidade. A incerteza estatística é resolvida através de observações. Por exemplo, quando uma moeda é lançada, tem-se a certeza de que o resultado será, antes de jogar a moeda, uma probabilidade de 50% para cada resultado.

No contexto da Lógica fuzzy então, uma incerteza, ou imprecisão, é uma propriedade inerente de um sistema e não pode ser alterada ou resolvida por observações, conforme afirma (ENGELBRECHT, 2007).

A Lógica Fuzzy permite o desenvolvimento de sistemas com capacidade de tornar qualquer solução mais intuitiva e natural, com menos regras e com maior legibilidade e facilidade de manutenção. Também torna possível que se desenvolva sistemas mais poderosos a partir da modelagem difícil de resolver usando técnicas tradicionais.

Lógica Fuzzy oferece as seguintes vantagens para os sistemas de desenvolvimento:

1. Permite tratar problemas não-lineares chamados “problemas difíceis”;

2. A modelagem cognitiva promove a melhoria entre o sistema perito ou o seu código e o problema subjacente e o processo de tomada de decisão capaz de representar vários especialistas em um problema, incluindo as regras conflitantes e tomada de decisão

3. Reduz codificação e complexidade em sistemas especialistas, exigindo menos regras, que também são mais facilmente compreensíveis;

4. Melhoram, de forma mais consistente, a modelagem matemática;

6 | CONCLUSÃO

Neste capítulo, abordou-se o referencial teórico sobre algoritmos de aproximação ou heurísticos, heurísticas, meta-heurísticas, hiper-heurísticas e lógica fuzzy. A revisão é necessária para se entender o problema a ser abordado no capítulo três, que trata do PRVC e sua intratabilidade por algoritmos exatos.

CAPÍTULO 3 - O ROTEAMENTO DE VEÍCULOS

1 | PRVC

O problema de roteamento de veículos é um clássico da otimização e pode ser enunciado conforme segue: dado um conjunto de clientes, cada um demandando por um produto e um depósito com veículos de capacidade Q, encontrar as rotas para todos os veículos minimizando os custos de transporte, como mostra a (Fig 3.1).

O PRV tem variantes e a escolhida no escopo desta pesquisa - a de veículos capacitados é a mais símples. Seu diferencial se dá pelo fato de que ela relaxa a restrição da limitação de tempo, não impondo penalidade ou limite quanto à duração máxima das rotas obtidas.

A principal restrição desta variante é que a soma da busca total de cada rota não pode exceder a capacidade do veículo destinado ao percurso, sendo que todos os veículos têm a mesma capacidade.

Dessa forma, o objetivo é minimizar o custo global, tanto do número de veículos envolvidos, quanto do tempo ou distância de viagem das rotas, levando em conta a capacidade de cada veículo.

Ao final, indica-se como devem ser alocados os veículos nas multiplas rotas determinadas tal que se realize a operação de distribuição com o menor custo possível.

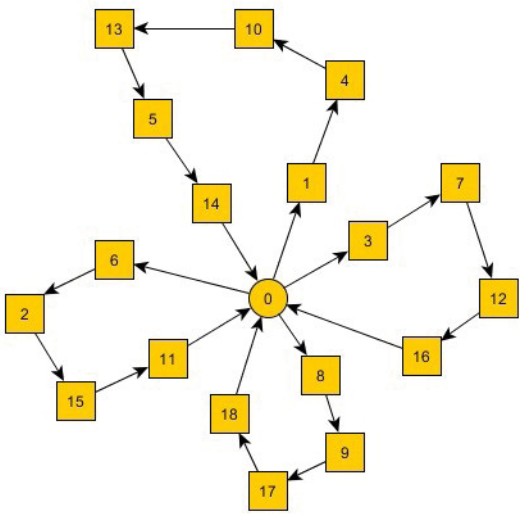


Figura 3.1: Problema do Roteamento de Veículos Capacitados

O PRV é uma generalização do Problema do Caixeiro Viajante (PCV) e pertence à classe de problemas NP-Difícil, não existindo algoritmos em tempo polinomial que encontre soluções exatas para ele.

Para contornar o problema, existem soluções por aproximação, utilizando-se para tal, algoritmos de uma classe especial da chamada otimização combinatória. Esses algoritmos têm evoluido bastante a partir do aperfeiçoamento dos recursos computacionais e do surgimento de soluções inspiradas na teoria de evolução, tendo-se registro de resultados acima da média com algoritmos genéticos, por exemplo.

1.1 Formulação

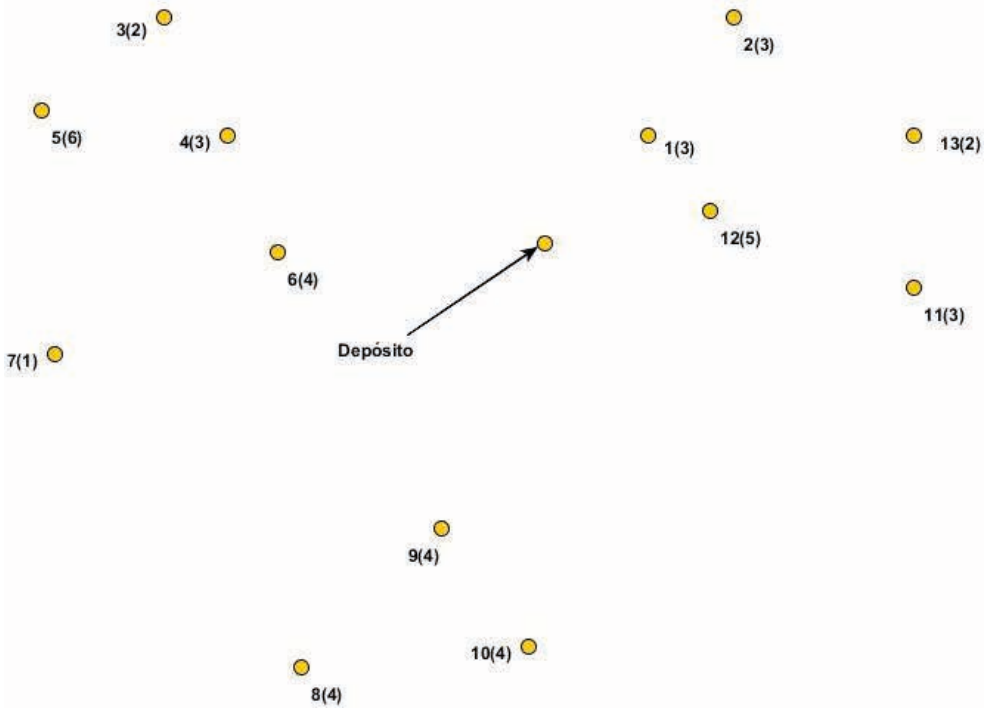
Dado um grafo G = (V, E) um grafo não direcionado, onde $V = v_0, v_1,..., v_n$ é o conjunto dos vértices e $E = (v_i, v_j)$: $v_i, v_j \in V, i < j$, o conjunto de arestas. O vértice v_0 representa o depósito, sendo esse vértice a base de uma frota de veículos de capacidade Q. Os vértices restantes representam os clientes. Cada cliente v_i tem uma demanda não negativa q_i e $q_0 = 0$.

Neste trabalho, utilizou-se instâncias com 25 e 250 veículos no depósito. A cada aresta (v_i, v_j) está associada uma distância não negativa c_{ij} que representa a distância entre os consumidores. Assim, o PRVC consiste em determinar um conjunto de rotas que devem ser cumpridas pelos veículos minimizando os custos de transporte, que é dado pela distância, levando em conta as seguintes restrições:

- Toda rota começa e termina no depósito.
- Toda cidade de V, com exceção do depósito, é visitada somente uma vez por somente um veículo.
- A demanda total de qualquer rota não deve superar a capacidade Q de um veículo.

1.2 Representação

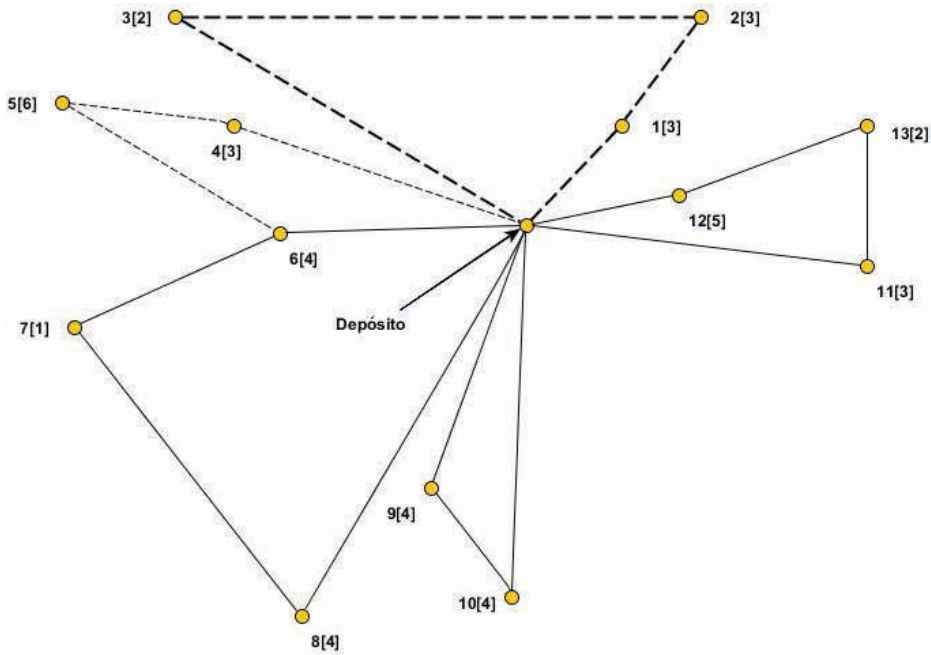
Uma solução genérica do PRV é representada por meio de uma permutação de cidades, numeradas de 1 a n, separadas em tantas partições quantos forem o número de veículos usados. O elemento separador é representado pelo valor zero e indica o depósito (Fig 3.2).



(a) Pet1

Figura 3.2: Exemplo de problema - Adaptado de (RYAN; HJORRING; GLOVER, 1993)

No mundo real, cada rota percorrida por um único caminhão é denominada pétala (Fig 3.3). No conceito de pétala, cada um dos pontos de entrega contém uma exigência de entrega associada mostrada entre colchetes. O objetivo é o de minimizar o número de veículos necessários para entregar a partir de um depósito central, minimizando a distância total percorrida (HJORRING, 1993).



(a) Pet2

Figura 3.3: Solução ótima baseada em pétala - Adaptado de (RYAN; HJORRING; GLOVER, 1993)

Os pontos de entrega são numerados em ordem radial sobre o depósito. Cada subconjunto radial contíguo obtido a partir dessa ordenação é uma pétala. Uma pétala é viável se a quantidade de bens entregues na rota não exceda a capacidade do veículo e se o total da distância percorrida, conforme determinado pela sequência de viagens das entregas não exceda o limite de distância imposta.

O custo de uma pétala é definido como sendo o custo da rota pétala correspondente, que é, tipicamente, embora não necessariamente, proporcional à distância percorrida.

Em 1992, (HJORRING; RYAN, 1992) discutem técnicas de caminho mais curto eficiente para produzir soluções ótimas em pétala. Nesse trabalho, apresentam uma nova heurística para o PRV. A solução apresentada usa busca tabu para conduzir um método em pétala. O algoritmo é iniciado com uma solução pétala e em seguida, produz tentativas de melhorar a solução considerando pequenas perturbações da ordem radial. O algoritmo de mínimos locais escapa através da utilização de um lista tabu. O efeito de diferentes critérios tabu e comprimentos de lista tabu são discutidos. Os testes conduzidos mostram que o método é competitivo com outras heurísticas de PRV de alta qualidade.

1.3 Formulação matemática

Para se avaliar as soluções do PRV, utiliza-se uma função objetivo, cuja aplicação sobre uma determinada solução s obtém o resultado, que é o somatório de todas as

distâncias percorridas por todos os veículos. A formulação matemática está baseada no trabalho de (FIGUEIREDO, 2007) e reflete um típico problema de planejamento de rotas de veículos.

Nesse trabalho, considera-se que o custo de visitar cada cliente, independe dos veículos que o fazem, sendo que não se consideram restrições que resultem na incompatibilidade entre veículos e clientes. O objetivo é encontrar m rotas em *G* que minimizem o custo e satisfaçam as restrições impostas.

Função objetivo

$$Minimizar \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{m} c_{ij}x_{ijk} \tag{3.1}$$

Restrições Sujeito a:

$$\sum_{i=0}^{n}\sum_{k=1}^{m} x_{ijk} = 1, \qquad j = 0,1,...,n \tag{3.2}$$

$$\sum_{j=1}^{n} x_{0jk} \leq 1, \qquad k = 1,...,m \tag{3.3}$$

$$\sum_{i=0}^{n} x_{ipk} = \sum_{j=0}^{n} x_{pjk}, \qquad k = 1,...,m; p = 0,...,n \tag{3.4}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} q_i x_{ijk} \leq Q_k, \qquad k = 1,...,m \tag{3.5}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} (t_{ij} + t_i)x_{ijk} \leq D_k, \qquad k = 1,...,m \tag{3.6}$$

$$\sum_{v_i \in S}\sum_{v_j \in S} x_{ijk} \leq |S| - 1, \qquad k = 1,...,m; 2 \leq |S| \leq V \tag{3.7}$$

$$x_{ijk} \in \{0,1\}, \qquad i,j = 1,...,n; k = 1,...,m \tag{3.8}$$

- A função objetivo (3.1) minimiza o custo total no grafo G.
- Em (3.2), é garantido que cada cliente será atendido por um único veículo
- Em (3.3), restringe-se ou não a saída do veículo k. Caso a operação seja necessária, ele realiza rota única
- Em (3.4), a equação garante a circulação restrita à rota, forçando determinado veículo k a visitar o cliente j, somente depois que visitar o cliente i.
- Em (3.5) e (3.6), declaram-se duas restrições - a primeira, determina a capacidade máxima do veículo. A segunda, o tempo máximo do percurso de cada veículo na rota determinada
- Em (3.7), restringe-se a existência de sub-rotas. (S é um sub-conjunto de clientes)
- Em (3.8), é definido o tipo de variáveis de decisão - Nesse caso, binárias

Na formulação, um conjunto de índices, parâmetros e variáveis compõem a estrutura de dados, como segue:

Índices

i	Local visitado
j	Local a visitar
k	Veículo

Parâmetros

c_{ij}	Custo de deslocamento entre o local i e o local j
q_i	Quantidade pedida pelo local i
Q_k	Capacidade máxima do veículo k
D_k	Duração máxima da rota percorrida pelo veículo k
t_{ij}	Tempo de viagem entre o ponto i e j
t_i	Tempo necessário na operação de descarga do local i

Variáveis de decisão

X_{ijk} Variável binária que recebe valor 1 se o veículo k visita o cliente *j* após ter visitado o cliente i. Caso contrário assume valor 0

2 | TRABALHOS RELACIONADOS AO PRV

O primeiro estudo sobre o PRV, data de 1959 e foi realizado por (DANTZIG; RAMSER, 1959). Na proposta para otimização de uma frota de veículos de entrega de gasolina, o autor sugere a realização de cálculos manuais para encontrar rotas ou o emprego de um dispositivo computacional - raro, à época. Essa proposta baseada em uma formulação de programação linear garante soluções próximas de um ótimo e basicamente fornece rotas mais curtas entre quaisquer dois pontos do sistema. O trabalho pioneiro, mesmo com resultados bastantes preliminares, porém promissores, serviu apenas aos objetivos da inovação, não tendo resultados práticos, de imediato.

Em 1976 (CHRISTOFIDES, 1976) publicou uma pesquisa que serviu de base para muitas propostas que vieram depois. Nesse trabalho, o autor classifica o problema, discute e amplia métodos exatos e aproximados para solução do PRV, além de fornecer alguns resultados sobre as propriedades de soluções viáveis que ajudam a reduzir o esforço computacional envolvido na solução dessa classe de problemas.

Mais adiante, no ano de 1992, (LAPORTE, 1992) ressalta a importancia do PRV para áreas de distribuição e logística. Já nessa época existia grande variedade de soluções e ampla literatura sobre o problema. Nesse trabalho o autor realiza um levantamento

completo dos principais algoritmos exatos e aproximados desenvolvidos para a PRV, em um nível adequado para um primeiro curso de pós-graduação em otimização combinatória.

No ano de 1999, (GEHRING; HOMBERGER, 1999) aborda o problema do PRV com janelas de tempo seguindo uma estratégia de duas fases. O problema de roteirização de veículos com janelas de tempo (PRVJT) é uma extensão do PRV com um depósito central.

A função objetivo do PRVJT considerada nesse trabalho combina a minimização do número de veículos segundo um critério primário e a distância total da viagem, como critério secundário. O processo para obtenção da solução se dá pela paralelização. Na primeira fase, o objetivo é a minimização do número de veículos. Na segunda fase a distância total é minimizada utilizando um algoritmo de busca tabu.

A paralelização desse procedimento híbrido sequencial adota um conceito de autonomia cooperativa, onde vários procedimentos autônomos que conduzem a solução sequenciais cooperam mediante o intercâmbio de soluções. No trabalho, (GEHRING; HOMBERGER, 1999) enfatiza que o intercâmbio de soluções pode levar a saltos correspondentes no espaço de solução apenas se forem respeitadas determinadas condições de aceitação. Dessa forma, um bom desempenho tanto da abordagem sequencial a paralela só pode ser homologado se comparado a resultados bastante conhecidos.

Ainda no ano de 1999, (HOMBERGER; GEHRING; others, 1999) publica um trabalho sobre PRVJT, seguindo as mesmas técnicas de (GEHRING; HOMBERGER, 1999), mas com abordagem evolutiva nas duas etapas da solução. Nesse trabalho, o autor conduziu experimentos em 58 problemas da literatura com tamanhos variados de pontos (clientes) e veículos, chegando à conclusão de que estratégias de evolução são eficazes quando o cenário trata de reduzido número de veículos e distância total de viagem.

3 | CONCLUSÃO

Neste capítulo, abordou-se o PRVC, uma variante do problema do roteamento de veículos - PRV, sua representação matemática, e trabalhos relacionados.

CAPÍTULO 4 - HIPER-HEURÍSTICA PROGRAMAÇÃO EVOLUTIVA COM CONTROLE ADAPTATIVO FUZZY

1 | HIPER-HEURÍSTICA PROGRAMAÇÃO EVOLUTIVA COM ADAPTAÇÃO DE BUSCA LOCAL

A hiper-heurística EPH foi escolhida para ser a base da presente pesquisa pela sua proposta - é baseada na abordagem de programação evolutiva e co-evolução e pelo seu ciclo de vida em estágio inicial, mas bem consistente, sendo nesse sentido, um produto bem situado no contexto das hiper-heurísticas.

A EPH utiliza o arcabouço HyFlex, uma estrutura de software projetada para permitir o desenvolvimento, teste e comparação iterativo de algoritmos de uso geral de busca heurística como as hiper-heurísticas. Para atingir esses objetivos a EPH aproveita o baixo acoplamento da HyFlex e seu conceito de decomposição de um algoritmo de busca heurística em duas partes principais:

1. Parte de propósito geral: o algoritmo ou hiper-heurística.
2. O problema específico da parte: fornecido pela estrutura HyFlex.

Na literatura sobre hiper-heurísticas, esse conceito é referido como barreira de domínio entre as heurísticas específicas do problema e a hiper-heurística. A HyFlex estende a estrutura conceitual da barreira de domínio, mantendo uma população - em vez de uma única solução incumbente - na camada de domínio do problema. Além disso, fornece uma variedade de heurísticas específicas do problema e operadores de pesquisa. Fornece também, uma boa variedade de problemas de otimização combinatória testados sobre dados de instâncias do mundo real.

O framework foi escrito em Java orientado a objetos, independente de plataforma e gerenciamento automático de memória. No seu nível mais alto, o framework consiste em apenas duas classes abstratas: ProblemDomain e HyperHeuristic.

Uma configuração trivial da EPH é montada como segue:

1. Uma memória de população de soluções configurável pelo utilizador, que pode ser mantida pela hiper-heurística através de métodos como **setMemorySize** e **copySolution**.

2. Uma rotina para inicializar aleatoriamente soluções, **initialiseSolution(i)**, onde i é o índice da matriz de soluções na memória.

3. Um conjunto de heurísticas específicas do problema, que são utilizadas para modificar soluções.

4. Esse conjunto pode ser chamado com o método applyHeuristic(i,j,k), onde i é o índice da heurística a ser chamada, j é o índice da solução na memória e k é o indice na memória, onde a solução resultante deve ser colocada.

A hiper-heurística adota quatro grupos de heurísticas para aplicar na resolução de cada domínio de problema, como segue:

- Heurísticas mutacionais ou de perturbação: realizam uma pequena mudança na solução, trocando, alterando, removendo, adicionando ou excluindo uma solução.
- Heurísticas ruin-recreate - destruição-construção: destrói parcialmente uma solução e a reconstrói ou a recria depois. Estas heurísticas podem ser consideradas como grandes estruturas de vizinhança. Elas podem incorporar heurísticas de construção específicas para reconstruir as soluções.
- Heurísticas de busca local ou hill-climbing: iterativamente fazem pequenas mudanças na solução, aceitando apenas soluções de não-deterioração, até que o local ideal seja encontrado ou uma condição de parada for atendida. Essas heurísticas diferem das heurísticas de mutação porque incorporam uma melhoria iterativa e garantem que uma solução não vai se deteriorar.
- Heurísticas Crossover: Pegam duas soluções, combina-as e retornam uma nova solução.

Específicamente para o PRVC, a EPH utiliza um conjunto de 12 heurísticas de baixo nível: 4 heurísticas de mutação, 2 heurísticas ruin-recreate, 4 heurísticas de busca local, e 2 heurísticas de crossover, mais 1 heurística de inicialização chamada **Randomised constructive heuristic.**

Na abordagem evolutiva da EPH, duas populações (Fig 4.1) evoluem durante o processo de encontrar uma solução - uma população de soluções e uma população de sequências heurísticas.

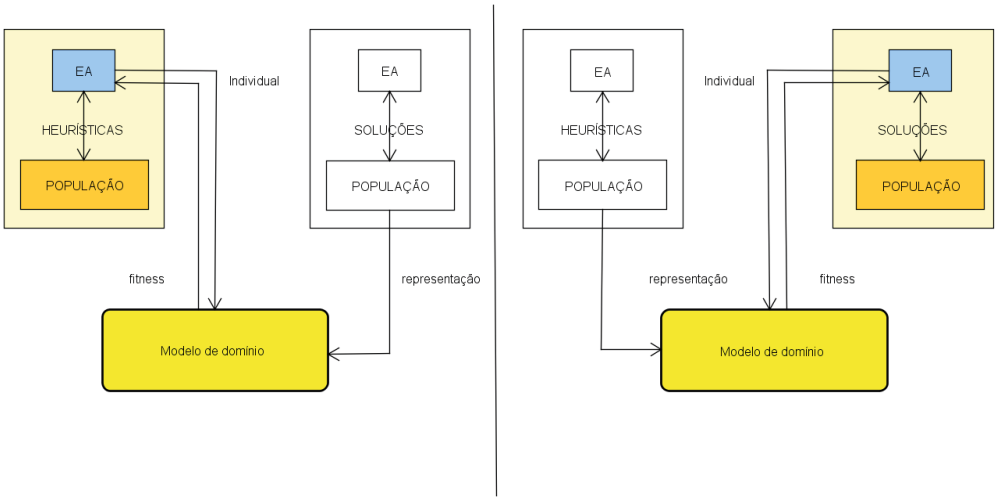


Figura 4.1: Co-evolução. Adaptado de POTTER; DE JONG (2000)

O conjunto de sequências heurísticas corresponde à estratégia de busca, que evolui através de mutação e seleção. A população de soluções e a população de heurísticas co-evoluem no sentido de que a evolução de soluções é feita pela aplicação de sequências heurísticas e evolução das sequências heurísticas diretamente relacionada à sua eficácia em soluções.

A estrutura e o processo relacionados com a população de soluções é simples. A população é constituída por um conjunto de N soluções. Tais soluções são usadas para avaliar sequências heurísticas. A avaliação de sequências de heurísticas é o processo-chave para produzir novas soluções.

Para avaliar uma sequência heurística, uma ou duas soluções são escolhidas na população. A seguir, a sequência de heurísticas é aplicada sobre essas soluções. Finalmente, a solução resultante é inserida na população caso satisfaça os critérios de aceitação.

O conjunto inicial de soluções é gerado usando uma heurística padrão de geração. A escolha das soluções para a avaliação de sequências heurísticas é randomizada - seguindo a ordem dos índices na população de soluções. Após a aplicação de uma sequência de heurísticas, a solução resultante é inserida na população se o seu custo é melhor do que, pelo menos, um custo na população de soluções e se esse custo for diferente de todos os custos na população de soluções.

Sempre que uma solução é inserida na população, esta substitui a pior solução. Assim, o tamanho da população permanece o mesmo. O conjunto de sequências das heurísticas corresponde à estratégia de busca que evolui durante o processo de resolução. A população é composta de um conjunto de S sequências heurísticas com base no mesmo padrão.

Uma sequência é composta por um conjunto de heurísticas de perturbação e um conjunto de heurísticas de busca local com um valor de intensidade associado para cada perturbação e heurística de busca local. A parte de perturbação contém uma ou duas heurísticas de perturbação: mutação, cruzamento ou destruir-recriar.

A fase de busca local contém todas as heurísticas de busca local disponíveis para o problema. A aplicação de uma sequência começa executando uma vez as heurísticas de perturbação, com os respectivos parâmetros de intensidade.

Uma heurística de perturbação consiste de um esquema de ações que ajudam a se evitar um local ótimo no espaço de busca. Logo que se inicia a execução da hiper-heurística, uma busca local é realizada a partir da solução inicial gerada. Na sequencia, o algoritmo realiza uma iteração, provocando modificações na posição atual, até que um critério de parada seja satisfeito.

Tais modificações são definidas como uma perturbação da solução atual. O objetivo disso é criar uma solução-candidata, que será intensificada, a seguir, por uma heurística de busca local. A solução-candidata intensificada é então verificada por um critério de

aceitação. Se aceita a solução, ela se torna a posição atual do algoritmo e será modificada na iteração seguinte.

Nos vários contextos dos problemas de distribuição, é necessário se construir um caminho curto a partir de um ponto com muitas coletas e/ou entregas. Tal como nos outros problemas de roteamento, no PRVC é muito difícil de resolver o fênomeno da otimalidade local, fazendo com que heurísticas de busca em muitas vezes fiquem presas em ótimos locais. As Heurísticas de Perturbação fornecem assim, um meio de escapar de ótimos locais.

Segundo (BLUM; ROLI, 2003), as heurísticas de perturbação têm uma importância vital na busca local. O autor afirma que uma perturbação muito pequena não permite que a heurística escape dos ótimos locais. Mas se uma perturbação for exagerada, isso a transformaria em uma heurística simples de busca local com reinício aleatório.

A Busca Local - também referenciada na literatura como busca na vizinhança, é uma estratégia básica de muitos dos métodos heurísticos empregados na solução de problemas de otimização combinatória.

Os algoritmos de busca local (Fig 4.2) comportam métodos de otimização iterativos. Iniciam com uma solução e iterativamente, procuram na vizinhança desta solução uma outra de menor custo ou que sinalize com uma possibilidade de ganho futuro. Se essa solução for encontrada, a solução inicial é substituída pela nova solução e a busca continua.

Iterated Local Search.

```
Input:
Output: S_best
1  S_best ← ConstructInitialSolution();
2  S_best ← LocalSearch();
3  SearchHistory ← S_best;
4  while ¬ StopCondition() do
5  |  S_candidate ← Perturbation(S_best, SearchHistory);
6  |  S_candidate ← LocalSearch(S_candidate);
7  |  SearchHistory ← S_candidate;
8  |  if AcceptanceCriterion(S_best, S_candidate, SearchHistory) then
9  |  |  S_best ← S_candidate;
10 |  end
11 end
12 return S_best;
```

Figura 4.2: Iterated Local Search - Pseudocódigo extraído de BROWNLEE (2011)

Dentre os métodos baseados em busca local os mais conhecidos e utilizados, hoje, são têmpera simulada, algoritmos genéticos, busca tabu e busca local dirigida.

Para (STÜTZLE, 1999), o histórico de uma busca local pode ser utilizado para

influenciar mecanismos de modificação de avaliação dos critérios de aceitação. Afirma ainda que essa é uma característica opcional das implementações da Busca Local.

Especificamente na EPH (MEIGNAN, 2011), quando uma heurística é aplicada, a solução resultante serve como solução inicial para a heurística seguinte na sequência.

Em seguida, buscas heurísticas locais são aplicadas uma vez, ou aplicado utilizando uma estratégia Descendente de Vizinhança Variavel (Variable Neighborhood Descend - VND) (HANSEN et al., 2010). Ambas as estratégias de busca local usam os parâmetros de intensidade e a ordem de heurísticas de busca local especificada pela sequência. A ação de escolha entre a VND (Fig 4.3) e aplicação original de heurísticas de busca local é determinada na fase de inicialização da hiper-heurística.

Algorithm 2 Variable neighborhood descent

Function VND (x, k_{max})
1 $k \leftarrow 1$
2 **repeat**
3 $\quad x' \leftarrow arg\min_{y \in N_k(x)} f(y)$ // Find the best neighbor in $N_k(x)$
4 $\quad x, k \leftarrow$ `NeighborhoodChange` (x, x', k) // Change neighborhood
until $k = k_{max}$
return x

Figura 4.3: VND - Pseudocódigo extraído de GENDREAU; POTVIN (2010)

A população inicial de sequências de heurísticas é gerada aleatoriamente. Essa população evolui até o final do processo de resolução, por mutação e seleção. Para obter uma nova geração, uma mutação é aplicada sobre cada sequência de heurísticas e dobra o tamanho da população. Depois, as sequências de heurísticas são selecionadas utilizando a seleção por torneio, que avalia sequências heurísticas para aplicá-la. A nova geração é composta por sequências de heurísticas vencedoras dos torneios.

Quatro operadores de mutação são definidos para sequências de heurísticas. Eles têm a mesma probabilidade de serem aplicados. A primeira mutação consiste em modificar todos valores de parâmetro de intensidade para a parte de perturbação da sequência. A segunda mutação acrescenta aleatoriamente, substitui ou remove uma heurística de perturbação enquanto limita o tamanho da parte de perturbação de uma ou duas heurísticas. A terceira mutação altera todos os valores de parâmetro de intensidade para a fase de sequência de busca. A última mutação altera aleatoriamente a ordem das autoridades locais heurísticas de busca. A seleção de sequências de heurística é feita na população de sequências parentes e adicionada às sequências mutadas. Cada torneio confronta duas sequencias escolhidas aleatoriamente. As sequências concorrentes são removidas da população inicial e as sequências vencedoras formam uma nova geração. O vencedor de um torneio é a sequência que tem escores mais altos em cada rodada. A rodada consiste em aplicar as duas sequências de uma mesma solução inicial - e uma

segunda solução para cruzamento.

Os parâmetros da solução EPH (MEIGNAN, 2011) são os seguintes:

- N - Tamanho da população de soluções,
- S - Tamanho da população de sequências de heurística,
- r - Número de rodadas para a seleção de sequências,
- O tipo de estratégia para a aplicação de heurísticas de busca local (VND).

O parâmetro r é fixado em 3, independentemente da instância a ser resolvida. O tamanho S da população de sequências é definida como o valor do número de heurísticas disponíveis para o problema. O tamanho da população de soluções e o tipo de estratégia para a busca local é determinado a partir de um perfil do problema. O perfil é um conjunto de valores que caracterizam a operação heurística no exemplo resolvido.

O objetivo desse perfil é analisar o comportamento de heurísticas na instância tratada. Perfil e valores são calculados em uma fase preliminar da hiper-heurística. Esta fase registra o tempo médio de cada heurística sobre 5 execuções e análise de 5 pesquisas locais. Essas amostras de pesquisas locais VND e os tempos médios permitem a determinação dos seguintes valores:

- Número estimado de aplicações heurísticas no limite de tempo;
- Número médio de melhorias de solução durante VND;
- Número estimado de aplicações VND no limite de tempo;
- Proporção de VND que falharam em produzir uma solução melhor do que uma solução randomica gerada.

A dimensão da população e do tipo de busca local é determinado a partir desses parâmetros. Se a estimativa do número de aplicações heurísticas é muito baixo, um tamanho pequeno da população de soluções será definido e uma estratégia de busca local com base na aplicação simples da heurística é usada.

Uma solução EPH (Fig 4.4) é construída a partir de 6 arrays contendo heurísticas de busca local, de perturbação, mutação, crossover, ruína-recriação e um conjunto classificado como “outras heurísticas”.

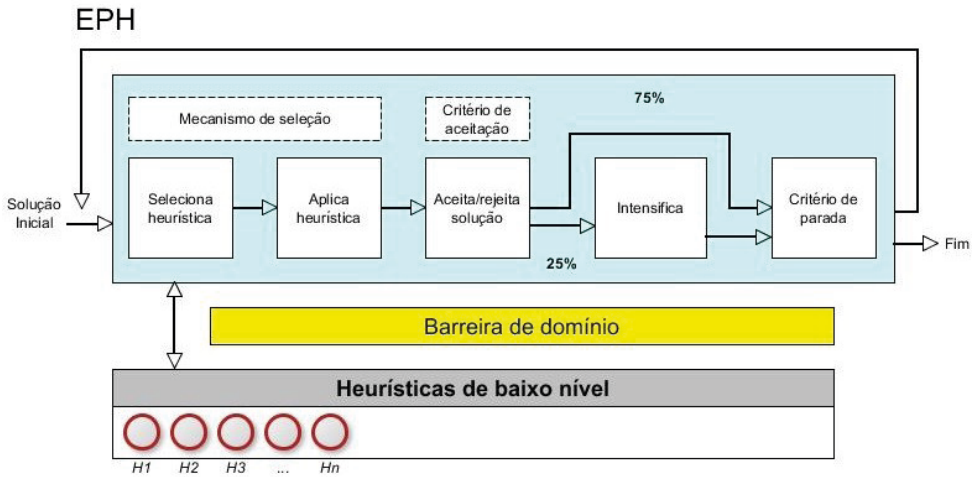


Figura 4.4: Configuração da EPH

Pela proposta do presente trabalho, um sétimo array passa a fazer parte da estrutura de dados da hiper-heurística. Essa estrutura na EPH_Fuzzy (Fig 4.5) tem a forma de uma janela circular Equação (5.1), que é avaliada a cada grupo de N melhores soluções. Dessa janela resulta a soma dos melhores resultados

$$\sum_{i=1}^{n} f(x_i) \;=\; x_1 + x_2 + x_3 \ldots + x_n \qquad (5.1)$$

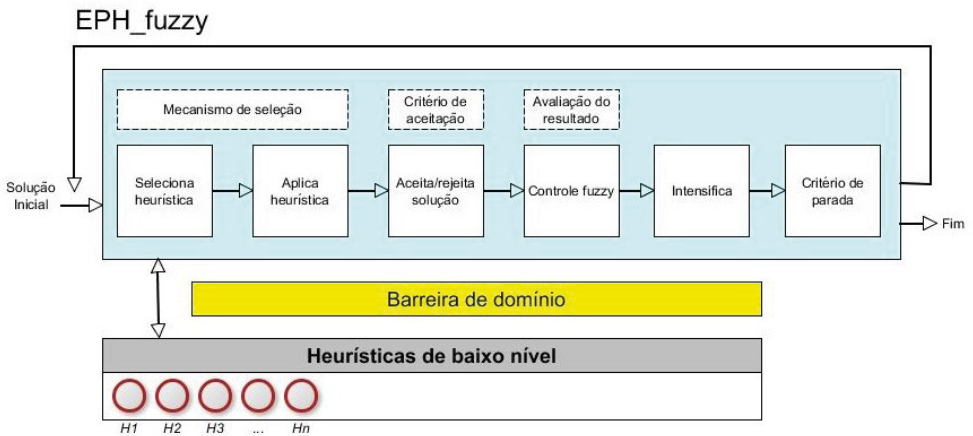


Figura 4.5: Configuração do Controle fuzzy

A partir da soma das soluções, a média, o melhor e o pior dos melhores resultados são passados ao avaliador de desempenho Equação (5.2)

$$desempenho = \frac{(media - melhor)}{(pior - melhor + 0.001)} \quad (5.2)$$

onde *media* é a média aritmética das últimas 10 soluções armazenadas na janela circular; *melhor* é a melhor solução presente na janela circular; e pior é a pior solução da janela circular. A constante 0.001 está presente na fórmula para corrigir o erro de divisão por zero quando pior e *melhor* forem iguais.

A equação (5.3) retorna o erro do desempenho.

$$erro.desempenho = \frac{(ref.desempenho - desempenho)}{ref.desempenho} \quad (5.3)$$

A equação (5.4) calcula a derivada do erro em relação a iteração anterior. Na primeira iteração a derivada será zero

$$derivada.desempenho = erro.desempenho - desempenho.anterior \quad (5.4)$$

A equação (5.5) passa cinco argumentos *Ke* (coeficiente de ganho proporcional) * *erro.desempenho*, *Kd* (coeficiente de ganho derivativo) * *derivada.desempenho* e três conjuntos: *sErro*, *sDeri*, e *sAcao* à uma função de controle, que retorna u.*desempenho*, que é o desempenho atual.

$$u.desempenho = controle(Ke * erro.desempenho, Kd * derivada.desempenho, sErro, sDeri, sAcao) \quad (5.5)$$

Na equação (5.6), o passo é determinado pela constante de ganho *Kup* = 1 multiplicada por *u.desempenho* que multiplica a diferença entre Limite inferior e o Limite superior do desempenho entre 0 e 1.

$$passo = KuP * u.desempenho * (DLL - DSS) \quad (5.6)$$

A equação (5.7) representa a profundidade, que é determinada por uma função de corte.

$$profundidade = corte(profundidade - passo, DSS, DLL) \quad (5.7)$$

Antes de se obter a saída do controlador, considera-se os elementos abaixo:

A variável desempenho é transmitida ao controlador fuzzy (Fig 4.6), juntamente com uma referência para o desempenho, arbitrada em 0.5. Esse fator é empírico e foi fixado com base em experimentos com taxas de 0.5, 0.7 e 0.9.

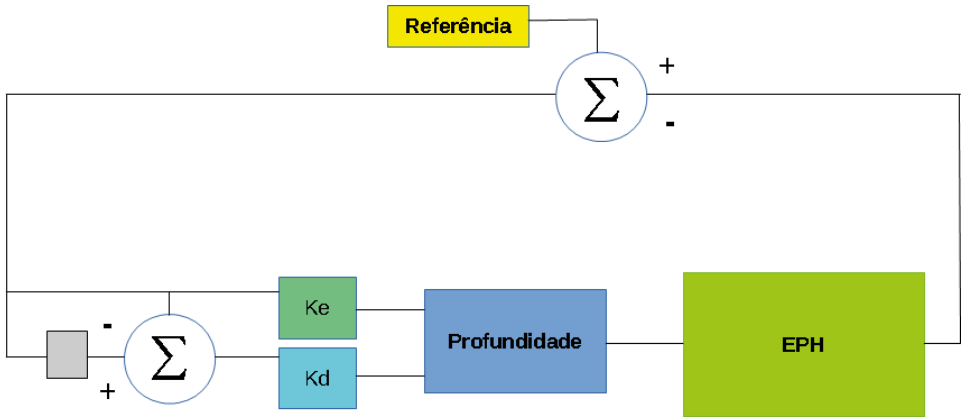


Figura 4.6: Controlador Fuzzy

A saída do controlador fuzzy é a profundidade, que acumula até 1., ao contrário da EPH, que considera a profundidade 1 somente após atingir 75% do tempo total ajustado para o algoritmo parar.

A taxa de mutação na EPH é aplicada pelas heurísticas de baixo nível de perturbação e é randomizada. Na presente proposta, essa taxa é definida em 0.5 até 33% do tempo total de execução, sendo levada a 1. quando o tempo total for > 33% até 66% e setada em 0. a partir de 66%

2 | O CONTROLADOR FUZZY PROPOSTO

Para o presente trabalho, adotou-se o modelo de Mamdani (Fig 4.7). Esse controlador é baseado em funções de implicação fuzzy e em operadores de composição para a definição da saída fuzzy do controlador. Nesse tipo de controlador a ação de controle é obtida por meio da definição de um conjunto de regras de controle fuzzy.

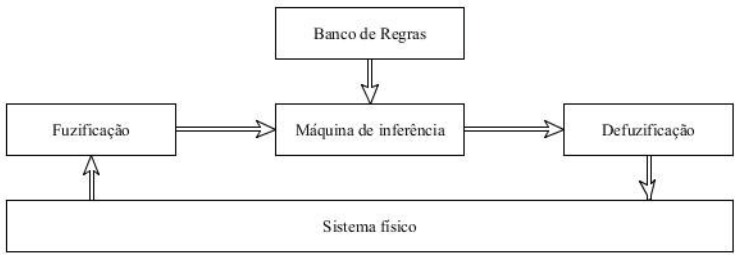


Figura 4.7: Esquema de um Controlador Mamdani

Essas regras são do tipo GMP, e uma dada regra (Ri) pode ser apresentada como:

Antecedente: x é *A'* e y é *B'*

Regra (R_i): se x é A_i e y é B_i então z é C_i

Consequente: z é C_i'

onde x and y são variáveis linguísticas que se relacionam ao estado do processo e z é a variável linguística de controle; $A', A_i, B', B_i, C'ieC_i$ são conjuntos fuzzy de x, y e z nos universos de dircurso U, V e W, respectivamente (ANDRADE; JACQUES, 2009)

O modelo Mamdani é baseado no trabalho publicado em 1974 por (MAMDANI, 1974). No algoritmo fuzzy do controlador, cada regra é uma proposição condicional fuzzy e diferentes relações fuzzy em $UxVxW$ podem se derivar dessas relações.

A implementação de cada regra é feita mediante a definição de operadores para o processamento do antecedente da regra e da função de implicação que irá definir o seu consequente. A ação do controlador fuzzy é definida pela agregação das n regras R_i que compõem o algoritmo, mediante o uso do conectivo "também", o qual pode ser implementado por diferentes operadores (ANDRADE; JACQUES, 2009).

Tal agregação resulta no conjunto fuzzy C, que define a saída do controlador C. Assim, a saída efetiva do controlador é obtida por um processo de defuzificação aplicado ao conjunto C.

Existem muitas possibilidades para se implementar os conectores das regras, das funções de implicação e do processo de defuzificação, sendo essas possibilidades bastante discutidas na literatura. O resultado da adoção de algumas dessas possibilidades e a definição dos conjuntos fuzzy associados às variáveis de entrada do modelo Mamdani foram objeto de pesquisas anteriores.

Na implementação do controlador, foi determinado uma base de regras comum (duas variáveis): onde 4 é um extremo (positivo grande) e 0 é outro (negativo grande)

3 | CONCLUSÃO

Neste capítulo, tratou-se da hiper-heurística EPH, suas particularidades, operação e sua relação com o controlador fuzzy implementado para atuar como parametrizador autônomo de profundidade de busca local, dando-se ênfase ao modelo Mamdani de controlador fuzzy adotado e o detalhamento da sua implementação.

CAPÍTULO 5 - EXPERIMENTOS COMPUTACIONAIS

O objetivo dos experimentos computacionais conduzidos na presente pesquisa é verificar a validade do controlador *fuzzy* para adaptar a intensidade da busca local através do ajuste do parâmetro de profundidade da EPH (CHESC 2011)[1] Os experimentos foram divididos em 2 fases. Na primeira fase, foram realizados testes com a EPH alterando-se empiricamente os parâmetros de profundidade, aplicando-se a intensificação da busca a partir de um ponto de corte, que em determinado momento, aconteceu a 25, 30 e depois a 50% do tempo de execução do algoritmo. Os melhores resultados foram obtidos a 25% do tempo de execução, como demonstrado na Tabela 5.1

Na segunda fase, com o controlador fuzzy implementado e devidamente calibrado, realizaram-se 3 experimentos, respectivamente com 0.5, 0.7, e 0.9 para a referência de desempenho.

Um problema específico foi escolhido para ser estudado - o Problema do Roteamento de Veículos, pelo desafio que representa e por existir farta documentação com muitos trabalhos publicados sobre o mesmo.

Nesse contexto, não se avaliou a EPH (MEIGNAN, 2011), nem tampouco o framework HyFlex com seu arcabouço e métodos e sim a atuação do controlador fuzzy e sobre a capacidade que o mesmo tem no processo de melhoria de uma solução computacional, não sendo aqui discutida a regularidade em produzir resultados satisfatórios em diversas instâncias e domínios de problemas como defende (BURKE et al., 2003a). Todas as instâncias foram executadas segundo a proposta original da EPH (MEIGNAN, 2011) - Aplicar o fator 1 à busca a partir de 75% do tempo decorrido, usando-se os resultados como base para comparar todos os resultados daí em diante. Ao longo da primeira fase surgiu uma alternativa à implementação original, que foi acrescentar um segundo cenário, onde a profundidade seria aplicada a partir da primeira solução, conforme valores gerados aleatoriamente entre 0.0 e 1.0 .

Instância	Nome	Veículos	Capacidade	AdOr-ILS	EPH
0	Solomon/RC/RC207	25	1000	5281.71	5057.06
1	Solomon/R/R101	25	200	21291.89	20650.80
2	Solomon/RC/RC103	25	200	13605.03	12317.03
3	Solomon/R/R201	25	1000	6564.42	5274.13
4	Solomon/R/R106	25	200	14280.79	13293.50
5	Homberger/C/C1-10-1	250	200	155305.46	146494.27
6	Homberger/RC/RC2-10-1	250	1000	77302.72	62110.19
7	Homberger/R/R1-10-1	250	200	163177.74	160227.01
8	Homberger/C/C1-10-8	250	200	158941.93	158349.25

1 http:www.asap.cs.nott.ac.ukexternalchesc2011

9	Homberger/RC/RC1-10-5	250	200	149447.68	147780.17

Tabela 5.1: EPH sem o controlador Fuzzy, com profundidade intensificada a 75% do tempo decorrido

Para conduzir os experimentos computacionais, escolheram-se 10 instâncias aleatórias do PRV a partir do repositório da HyFlex. O método, mesmo empírico, representa bem, um conjunto para o PRV, pois as instâncias Solomon, e Homberger[2] são muito conhecidas e bem documentadas.

1 | HARDWARE UTILIZADO

Os experimentos foram realizados em microcomputador Intel Core I5[3], de 2.4 GHz, 8GB de memória RAM. A implementação de melhorias da EPH se deu sob sistema operacional Windows 7, ambiente Eclipse Luna[4] com Java 8, e consistiu basicamente na implementação de dois dispositivos - um de avaliação de desempenho e um controlador fuzzy para atuar de forma autonoma na definição da profundidade de busca local. Ambos foram implementados em Java 8[5]. O conjunto de rotinas executou sob JVM 8 1.8 na versão de 64 bits.

Nenhum aditivo foi inserido no código, nem tampouco bibliotecas adicionais. O código da EPH (MEIGNAN, 2011) é público, e pode ser obtido em[6]. O código referente ao desempenho foi inteiramente desenvolvido durante a pesquisa. O código do controlador fuzzy foi traduzido do original, escrito em c, por[7] para Java.

Foram estabelecidas as seguintes regras:

- 30 execuções para cada instância
- Critério de parada de 120 segundos
- 60 resultados, obtidos a cada 2 segundos

Para finalidade de comparação dos resultados, apenas a última coluna de cada linha da matriz 30X60 de resultados foi selecionada e posteriormente inserida em uma planilha MINITAB[8] para se obter os boxplot's das mesmas. O boxplot[9] ou gráfico de caixa é muito empregado para avaliar a distribuição empírica de dados, sendo composto de:

1. Valor mínimo
2. Primeiro quartil (Q1)
3. Mediana (segundo quartil Q2)

2 https:www.sintef.noprojectwebtopvrptw
3 http:www.intel.com
4 http:www.eclipse.org
5 http:java.compt_BR
6 https:github.comdmeignaneph-chesc
7 http:www.deinf.ufma.br acmo
8 http:www.minitab.com
9 http:objetoseducacionais2.mec.gov.br

4. Terceiro quartil (Q3)

5. Distância Interquartil (IQ)

6. Desvio padrão

7. Valor máximo

Assim, no contexto desta pesquisa:

- O Valor Mínimo diz respeito à melhor solução encontrada na amostra
- A mediana (ou segundo quartil) é resultante da média entre as duas soluções centrais da amostra
- O primeiro quartil representa a faixa de soluções abaixo da mediana
- O terceiro quartil representa a faixa de soluções acima da mediana
- A distância interquartil é a diferença entre Q3 e Q1
- O desvio padrão demonstra o quanto as soluções se aproximam ou se distanciam da média
- Além das informações acima, os boxplots demonstram soluções discrepantes, acima da faixa de resultados confiável e soluções desprezíveis, abaixo do primeiro quartil.

Para detalhar os resultados, tabelas descritivas dos gráficos foram providenciadas e constam neste trabalho.

2 | RESULTADOS OBTIDOS

Para referência de desempenho 0.5, todas as instâncias tiveram menor dispersão. Em instâncias difíceis, observa-se a existencia de números discrepantes. Para instâncias pequenas, a EPH apresentou resultados bons. Mas para as instâncias maiores, mostrou valores atípicos elevados.

O melhor resultado geral na instância R208 (Fig 5.1) foi obtido pela EPH_Fuzzy, conforme números da Tabela 5.2. Observa-se que esta instância tem mediana assimétrica, apresentando uma faixa menor de melhores soluções. O desvio padrão demonstra uma alta diversidade de melhores soluções - espalhadas. Apesar de apresentar a pior solução máxima dentre as 3 obtidas com o controlador fuzzy, observa-se que as diferenças não são significativas.

Figura 5.1: Solomon_100 - R208

Parâmetro	Média	Dist.Iq	Desvio	Mínimo	1.Quartil	Mediana	3.Quartil	Máximo
EPH	3790,40	39,30	173,10	2883,30	3801,70	3820,60	3841,00	3870,60
EPH_RAND	3826,80	37,20	22,20	3790,20	3808,80	3827,40	3846,00	3869,40
0.5	3760,80	40,40	243,80	2866,80	3796,70	3826,50	3837,10	3863,50
0.7	3783,00	34,30	171,40	2882,80	3794,40	3815,50	3828,70	3861,10
0.9	3786,50	25,90	168,80	2900,00	3800,80	3813,00	3826,70	3860,30

Tabela 5.2: R208

Na instância RC207 (Fig 5.2), o melhor resultado foi obtido pela EPH_Fuzzy, conforme Tabela 5.3. Neste caso, obteve-se a menor distância interquartil, a melhor média de soluções e o pior máximo dentre os resultados obtidos com o controlador. Esta instância tem uma mediana assimétrica, apresentando uma faixa menor de melhores soluções. O desvio padrão é o mais alto entre todas as soluções. Apresenta a pior solução máxima dentre as 3 obtidas com o controlador fuzzy, mas as diferenças não são significativas e não tem números discrepantes.

Figura 5.2: Solomon_100 - RC207

Parâmetro	Média	Dist.Iq	Desvio	Mínimo	1.Quartil	Mediana	3.Quartil	Máximo
EPH	5131,50	37,60	31,00	5075,70	5110,20	5127,90	5147,80	5200,60
EPH_RAND	5107,40	43,80	147,60	4344,60	5105,40	5131,00	5149,20	5247,90
0.5	5099,80	35,10	168,60	4218,90	5110,90	5129,60	5146,00	5188,10
0.7	5132,20	42,80	33,30	5056,30	5113,00	5127,90	5155,80	5208,30
0.9	5126,80	46,40	25,70	5089,20	5104,00	5125,00	5150,40	5170,80

Tabela 5.3: RC207

Conforme verificado durante os experimentos, o conjunto [EPH,Controlador fuzzy] se saiu melhor em instâncias difíceis. No caso da instância RC1_10_10 (Fig 5.3), o resultado melhor também foi obtido pela EPH_Fuzzy, como mostra Tabela 5.4. Para esta instância, obteve-se a melhor distância interquartil, o menor desvio padrão, a menor média de resultados e o menor dentre os resultados piores. Esta instância tem uma mediana quase simétrica, apresentando uma faixa equilibrada de melhores soluções. Utilizando o padrão de referência de desempenho = 0.5 foi possível se observar um equilíbrio nos resultados, não apresentando valores desprezíveis, abaixo do primeiro quartil e nem valores discrepantes, acima do terceiro quartil.

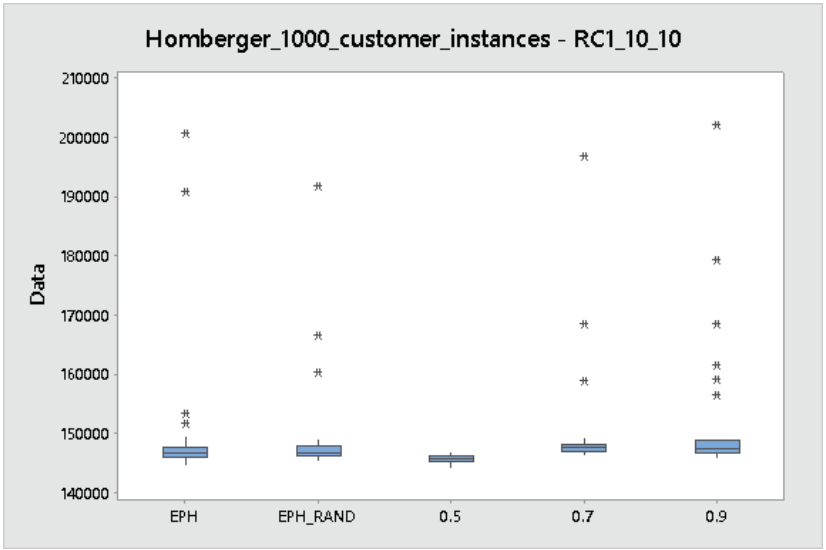


Figura 5.3: Homberger_1000_customers_instance - RC1_10_10

Parâmetro	Média	Dist.Iq	Desvio	Mínimo	1.Quartil	Mediana	3.Quartil	Máximo
EPH	150391,00	1759,00	12595,00	144644,00	146074,00	146810,00	147833,00	200906,00
EPH_RAND	149571,00	1738,00	9113,00	145339,00	146280,00	146819,00	148018,00	191860,00
0.5	145845,00	945,00	717,00	144147,00	145431,00	145918,00	146376,00	146940,00
0.7	150324,00	994,00	9830,00	146256,00	147108,00	147760,00	148102,00	196923,00
0.9	152049,00	2136,00	12141,00	145850,00	146773,00	147431,00	148909,00	202209,00

Tabela 5.4: RC1_10_10

O conjunto [EPH,Controlador fuzzy] nesta instância também se saiu melhor (Fig 5.4). Como mostra a Tabela 5.5, obteve a melhor distância interquartil, o menor desvio padrão, a menor média de resultados e o menor dentre os resultados piores. Esta instância tem uma mediana quase simétrica, apresentando uma faixa equilibrada de melhores soluções. Utilizando o padrão de referência de desempenho = 0.5 foi possível se observar um equilíbrio nos resultados, não apresentando valores desprezíveis abaixo do primeiro quartil.

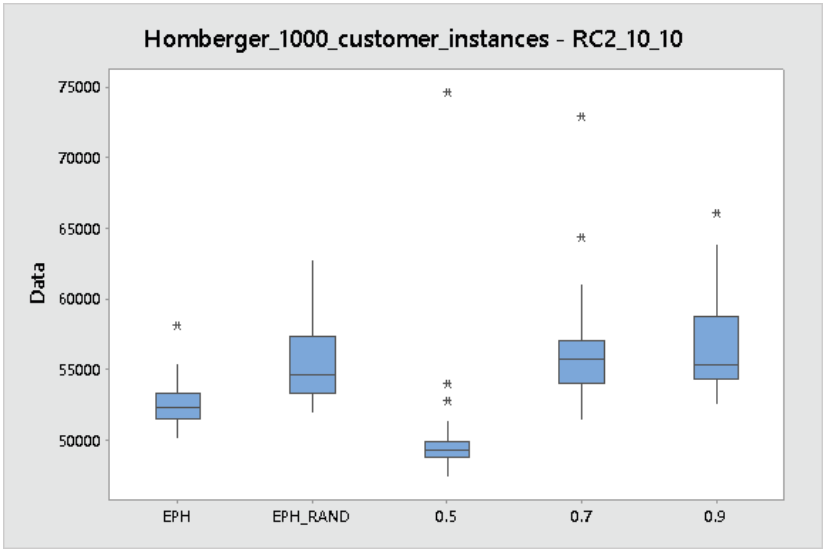


Figura 5.4: Homberger_1000_customers_instance - RC2_10_10

Parâmetro	Média	Dist.Iq	Desvio	Mínimo	1.Quartil	Mediana	3.Quartil	Máximo
EPH	52596,00	1870,00	1561,00	50053,00	51466,00	52335,00	53336,00	58197,00
EPH_RAND	55451,00	4053,00	2705,00	51941,00	53273,00	54665,00	57326,00	62798,00
0.5	50330,00	1126,00	4692,00	47404,00	48810,00	49307,00	49936,00	74634,00
0.7	56417,00	3016,00	4134,00	51371,00	53985,00	55758,00	57001,00	72959,00
0.9	56689,00	4419,00	3267,00	52552,00	54343,00	55339,00	58762,00	66080,00

Tabela 5.5: RC2_10_10

Esta instância apresentou a pior média com o controlador fuzzy parametrizado com 0.5 de referência de desempenho. Mas apresentou o melhor resultado como mostra a Figura 5.5, com um desvio padrão alto e valores fora da faixa do terceiro quartil. Dentre os resultados avaliados, também apresentou o maior pico de resultado (Tabela 5.6), com soluções acima do terceiro quartil. Não foi possível investigar esse desempenho, ficando para uma nova bateria de experimentos em futuro próximo.

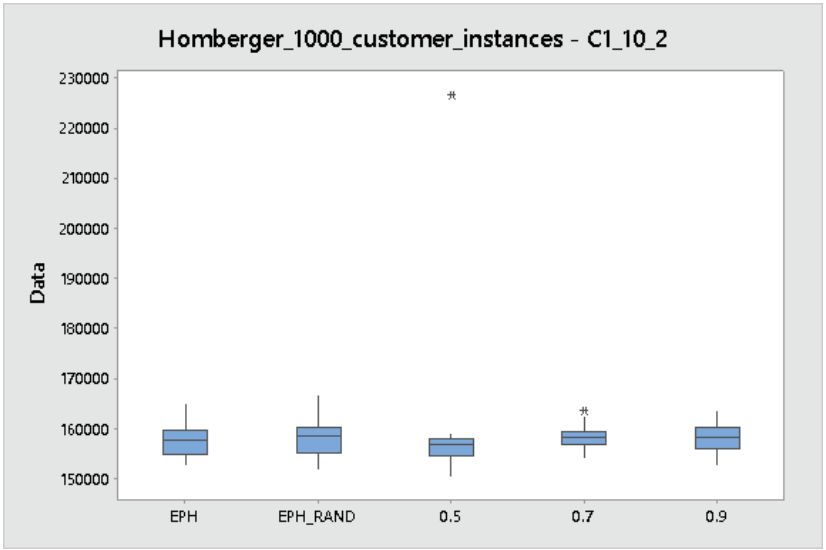


Figura 5.5: Homberger_1000_customers_instance - C1_10_2

Parâmetro	Média	Dist.Iq	Desvio	Mínimo	1.Quartil	Mediana	3.Quartil	Máximo
EPH	157676,00	4874,00	2940,00	152613,00	154944,00	157661,00	159818,00	165177,00
EPH_RAND	158314,00	5205,00	3430,00	151869,00	155088,00	158439,00	160293,00	166694,00
0.5	158685,00	3552,00	12992,00	150375,00	154514,00	156900,00	158066,00	226576,00
0.7	158271,00	2415,00	2295,00	153906,00	156913,00	158410,00	159328,00	163757,00
0.9	158040,00	4297,00	2789,00	152619,00	155962,00	158392,00	160259,00	163646,00

Tabela 5.6: C1_10_2

A instância (Fig 5.6) apresentou a menor distância interquartil, o menor desvio padrão, a melhor média e o melhor resultado com o controlador fuzzy parametrizado com 0.5 de referência de desempenho (Tabela 5.7).

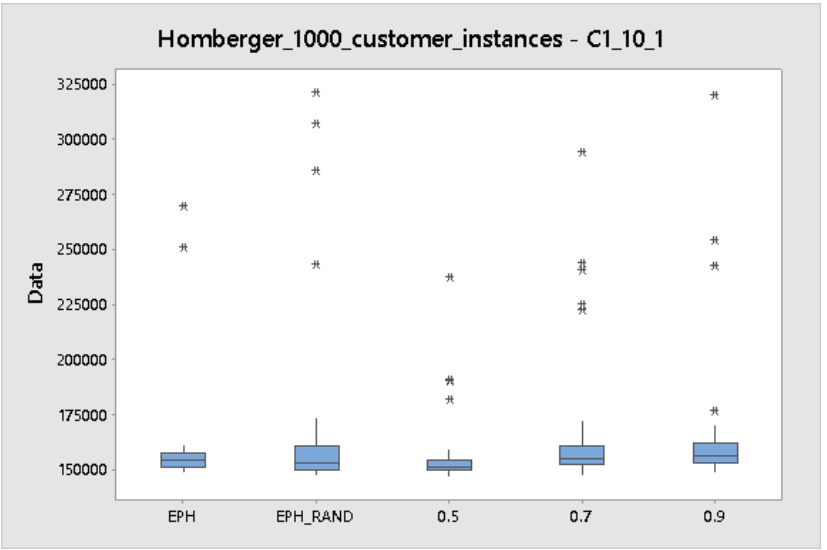


Figura 5.6: Homberger_1000_customers_instance - C1_10_1

Parâmetro	Média	Dist.Iq	Desvio	Mínimo	1.Quartil	Mediana	3.Quartil	Máximo
EPH	161246,00	6338,00	27388,00	148377,00	151454,00	154503,00	157792,00	270056,00
EPH_RAND	171804,00	11299,00	48508,00	147083,00	149560,00	153025,00	160859,00	321268,00
0.5	157706,00	4642,00	18962,00	146794,00	149744,00	151223,00	154386,00	237319,00
0.7	170102,00	8208,00	36250,00	146997,00	152719,00	154892,00	160927,00	294098,00
0.9	168601,00	9261,00	37332,00	148790,00	152995,00	156224,00	162256,00	319809,00

Tabela 5.7: C1_10_1

3 | CONCLUSÃO

Demonstrou-se neste capítulo os resultados obtidos nos experimentos realizados ao longo da presente pesquisa e a importância da proposta apresentada.

CAPÍTULO 6 - CONCLUSÃO

1 | CONSIDERAÇÕES FINAIS

Sem pretensão de ser conclusiva a afirmação de que é possível fazer funcionar uma hiper- heurística junto com um controlador fuzzy, esta pesquisa pôde constatar a eficácia de um dipositivo para avaliar soluções obtidas a partir de um algoritmo fornecendo elementos para que um controlador fuzzy regule a parametrização da profundidade de busca local desse algoritmo.

Assim, a contribuição mais importante deste trabalho passa pela proposta de fazer um controlador fuzzy trabalhar em conjunto com uma hiper-heurística, atuando esse controlador sobre o parâmetro de profundidade da busca local.

A literatura sobre hiper-heurísticas é bem ampla, mas pouco tem-se abordado sobre co-evolução associada às hiper-heurísticas. A Hyflex tem essa característica, o que a torna muito promissora, podendo-se melhorá-la, mediante a adoção de uma abordagem fuzzy, como foi feito.

É importante ressalvar que os resultados obtidos nos experimentos se mostraram satisfatórios quando comparados à EPH, mas o raciocínio foi o de não se comparar resultados com a literatura, mas sim com a EPH, e mostrar que é possível a melhoria da mesma, assim como de qualquer outra hiper-heurística que tenha o mesmo expertise. Embora existam soluções superiores à hiper-heurística estudada, foi evitado o confronto. Inclusive porque limitou-se a um único domínio de problema.

O framework Hyflex tem se destacado bastante nos últimos anos, mas é um produto caixa-preta, restringindo o acesso ao seu código-fonte, o que impossibilita melhorias/ajustes no seu núcleo composto de heurísticas de baixo nível. Durante a fase de experimentos, tentou-se adicionar um conjunto de instâncias do PRV fora do contexto Solomon/Homberger que tornasse os resultados mais amplos, mas isso não foi possível. O modelo de carga das instâncias inviabilizou o processo. Assim, restringiu-se o limite dessas instâncias a aquilo que o framework dispõe.

Pretende-se em futuro próximo, implementar um conjunto de heurísticas de baixo nível inspirado na Hyflex , mas com uma visão mais aberta, que permita a melhoria constante do conjunto por quem se proponha a colaborar, refletindo assim um ciclo de vida mais longo.

Por fim, existe a ideia de integrar um avaliador de desempenho e controlador fuzzy, junto a um framework inteiramente brasileiro e de código aberto.

REFERÊNCIAS

ANDRADE, M.; JACQUES, M. A. P. Estudo comparativo de controladores de Mamdani e Sugeno para controle de tráfego em interseções isoladas. TRANSPORTES, Brasil, v.16, n.2, 2009.

BERNHARD, H.; KORTE, B.; VYGEN, J. Combinatorial optimization: theory and algorithms. Alemanha: Heidelberg: Springer-Verlag, 2008.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys (CSUR), Estados Unidos, v.35, n.3, p.268–308, 2003.

BROWNLEE, J. Clever algorithms: nature-inspired programming recipes. Austrália: Jason Brownlee, 2011.

BURKE, E.; KENDALL, G.; NEWALL, J.; HART, E.; ROSS, P.; SCHULENBURG, S. Hyper-heuristics: An emerging direction in modern search technology. In: Hand- book of metaheuristics. Estados Unidos: Springer, 2003. p.457–474.

BURKE, E.; KENDALL, G.; NEWALL, J.; HART, E.; ROSS, P.; SCHULENBURG, S. Hyper-heuristics: an emerging direction in modern search technology. International series in operations research and management science, Estados Unidos, p.457–474, 2003.

CHRISTOFIDES, N. The vehicle routing problem. RAIRO - Operations Research - Recherche Opérationnelle, França, v.10, n.V1, p.55–70, 1976.

DANTZIG, G. B. Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation. , Estados Unidos, 1951.

DANTZIG, G.; RAMSER, R. The Truck Dispatching Problem. Management Science 6, Estados Unidos, p.80–91, 1959.

ENGELBRECHT, A. P. Computational intelligence: an introduction. Inglaterra: John Wiley & Sons, 2007.

FERREIRA, A. B. d. H. Dicionário Aurelio da lingua portuguesa. 5.ed. Brasil: Editora Positivo, 2010. v.1.

FIGUEIREDO, F. Planeamento de Rotas de Veículos com Entregas Fraccionadas. 2007. Tese (Doutorado em Ciência da Computação) — Dissertação de Mestrado. Universidade Técnica de Lisboa, Instituto Superior Técnico.

GASPAR-CUNHA, A.; TAKAHASHI, R.; CARLOS HENGGELER, A. Manual de Computação Evolutiva e Metaheurística. 1.ed. Portugal: Imprensa da Universidade de Coimbra/Editora da Universidade Federal de Minas Gerais, 2012. v.1.

GATHERCOLE, C.; ROSS, P. et al. Small populations over many generations can beat large populations over few generations in genetic programming. Genetic programming, Estados Unidos, v.97, 1997.

GEHRING, H.; HOMBERGER, J. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: EUROGEN99, 1999. Proceedings. . . Springer Berlin, 1999. v.2, p.57–64.

GENDREAU, M.; POTVIN, J.-Y. Handbook of metaheuristics. Estados Unidos: Springer, 2010. v.2.

GLOVER, F.; KOCHENBERGER, G. A. Handbook of metaheuristics. Estados Unidos: Springer, 2003.

GRATCH, J.; CHIEN, S.; DEJONG, G. Learning search control knowledge for deep space network scheduling. In: TENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 1993. Proceedings. . . UMASS, 1993. p.135–142.

HANSEN, P.; MLADENOVIĆ, N.; BRIMBERG, J.; PÉREZ, J. A. M. Variable neighborhood search. Estados Unidos: Springer, 2010.

HJORRING, C. Using Genetic Algorithms and the Petal Method to Solve Vehicle Routing Problems. In: ANNUAL CONFERENCE OF THE OPERATIONAL RESEARCH SOCIETY OF NEW ZEALAND, 29., 1993. Anais. ResearchGate, 1993.

HJORRING, C.; RYAN, D. M. Using Tabu Search and the Petal Method to Solve Vehicle Routing Problems. In: ANNUAL CONFERENCE OF THE OPERATIONAL RESEARCH SOCIETY OF NEW ZEALAND, 28., 1992. Anais. ResearchGate, 1992.

HOAISS, A. Novo Dicionário Houaiss da lingua portuguesa. 1.ed. Brasil: Editora Objetiva, 2009. v.1.

HOMBERGER, J.; GEHRING, H.; others. Two evolutionary metaheuristics for the vehicle routing problem with time windows. Infor-Information Systems and Operational Research, Alemanha, v.37, n.3, p.297–318, 1999.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling sales- man problem. Proceedings of the American Mathematical society, Estados Unidos, v.7, n.1, p.48–50, 1956.

LAPORTE, G. The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research, Canadá, v.59, n.3, p.345–358, 1992.

LIMA, I.; PINHEIRO, C.; SANTOS OLIVEIRA, F. Inteligência artificial. Brasil: Elsevier Brasil, 2004. v.1.

LUKE, S. Essentials of metaheuristics. Inglaterra: Lulu Raleigh, 2009. v.3.

MAMDANI, E. H. Application of fuzzy algorithms for control of simple dynamic plant. In: INSTITUTION OF ELECTRICAL ENGINEERS, 1974. Proceedings. IET, 1974. v.121, n.12, p.1585–1588.

MEIGNAN, D. An Evolutionary Programming Hyper-heuristic with Co-evolution for CHeSC'11. CHeSC 2011, Inglaterra, 2011.

MICHALEWICZ, Z.; FOGEL, D. B. How to solve it: modern heuristics. Estados Unidos: Springer Science & Business Media, 2013.

PASCHOS, V. T. Concepts of Combinatorial Optimization. Estados Unidos: John Wiley & Sons, 2014.

PEARL, J. Heuristics. Estados Unidos: Addison-Wesley Publishing Company Reading, Massachusetts, 1984.

PEARL, J. Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, Estados Unidos, 1984.

POTTER, M. A.; DE JONG, K. A. Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evolutionary computation, Estados Unidos, v.8, n.1, p.1–29, 2000.

REBENNACK, S. Stable set problem: branch & cut algorithms stable set problem: branch & cut algorithms. In: Encyclopedia of Optimization. Estados Unidos: Springer, 2009. p.3676–3688.

ROMANYCIA, M. H.; PELLETIER, F. J. What is a heuristic? Computational Intelligence, Estados Unidos, v.1, n.1, p.47–58, 1985.

ROSS, P. Hyper-heuristics. In: Search methodologies. Estados Unidos: Springer, 2005. p.529–556.

RYAN, D. M.; HJORRING, C.; GLOVER, F. Extensions of the petal method for vehicle routeing. Journal of the Operational Research Society, Inglaterra, p.289–296, 1993.

STÜTZLE, T. G. Local search algorithms for combinatorial problems: analysis, improvements, and new applications. Alemanha: Infix Sankt Augustin, Germany, 1999. v.220.

TALBI, E.-G. Metaheuristics: from design to implementation. Estados Unidos: John Wiley & Sons, 2009. v.74.

TERASHIMA-MARÍN, H.; ROSS, P. Evolution of constraint satisfaction strategies in examination timetabling. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE (GECCO99, 1999, Estados Unidos. Proceedings. Citeseer, 1999.

TSANG, E.; VOUDOURIS, C. Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Operations Research Letters, Inglaterra, v.20, n.3, p.119–127, 1997.

TVERSKY, A.; KAHNEMAN, D. Judgment under uncertainty: heuristics and biases. science, Estados Unidos, v.185, n.4157, p.1124–1131, 1974.

WONG, R. T. Combinatorial Optimization: Algorithms and Complexity (Christos H. Papadimitriou and Kenneth Steiglitz). SIAM Review, Estados Unidos, v.25, n.3, p.424–425, 1983.

ZADEH, L. A. Information and control. Fuzzy sets, Estados Unidos, v.8, n.3, p.338– 353, 1965.

ANTONIO CLERTON SANTANA DE ARAUJO - O autor possui graduação em Análise e Desenvolvimento de Sistemas pela Fundação Universidade do Tocantins e mestrado em Ciência da Computação pela Universidade Federal do Maranhão. Especialista em Engenharia de Software pela Universidade Gama Filho e Docência do Ensino Superior pela Faculdade Faveni. Tem experiência na área de Ciência da Computação, com ênfase em Otimização combinatória, Engenharia de Software e Banco de Dados, atuando como desenvolvedor de sistemas e como docente em tecnologias da informação e da comunicação, sistemas tutoriais inteligentes, e software educacional. Professor da Universidade Estadual do Tocantins.

Uma abordagem hiper-heurística para problemas de otimização

www.atenaeditora.com.br
contato@atenaeditora.com.br
@atenaeditora
www.facebook.com/atenaeditora.com.br

Atena Editora
Ano 2023

Uma abordagem hiper-heurística para problemas de otimização

www.atenaeditora.com.br
contato@atenaeditora.com.br
@atenaeditora
www.facebook.com/atenaeditora.com.br

Atena Editora
Ano 2023