Um Estudo Empírico de Hiper-Heurísticas

Igor Ribeiro Sucupira

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Área de Concentração: Ciência da Computação.

Orientador: Prof. Dr. Flávio Soares Corrêa da Silva.

Durante a elaboração deste trabalho, o autor recebeu o auxílio financeiro do CNPq.

São Paulo, junho de 2007.

Um Estudo Empírico de Hiper-Heurísticas

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Igor Ribeiro Sucupira e aprovada pela Comissão Julgadora.

São Paulo, 3 de julho de 2007.

Banca Examinadora:

Prof. Dr. Flávio Soares Corrêa da Silva (orientador) – IME/USP.

Prof. Dr. José Augusto Ramos Soares – IME/USP.

Prof. Dr. Marco Túlio Carvalho de Andrade – EP/USP.

Resumo

Uma hiper-heurística [53] é uma heurística que pode ser utilizada para lidar com qualquer problema de otimização, desde que a ela sejam fornecidos alguns parâmetros, como estruturas e abstrações, relacionados ao problema considerado. As hiper-heurísticas têm sido aplicadas a alguns problemas práticos e apresentadas [57, 2, 9, 14, 16, 17, 15, 8, 13, 34, 32, 33, 59] como métodos de grande potencial, no que diz respeito à capacidade de possibilitar o desenvolvimento, em tempo bastante reduzido, de algoritmos capazes de lidar satisfatoriamente, do ponto de vista prático, com problemas de otimização complexos e pouco conhecidos. No entanto, é difícil situar as hiper-heurísticas em algum nível de qualidade e avaliar a robustez dessas abordagens caso não as apliquemos a problemas para os quais existam diversas instâncias disponíveis publicamente e já experimentadas por algoritmos relevantes. Este trabalho procura dar alguns passos importantes rumo a essas avaliações, além de ampliar o conjunto das hiper-heurísticas, compreender o impacto de algumas alternativas naturais de desenvolvimento e estabelecer comparações entre os resultados obtidos por diferentes métodos, o que ainda nos permite confrontar as duas diferentes classes de hiper-heurísticas que identificamos. Com essas finalidades em mente, desenvolvemos 3 novas hiper-heurísticas e implementamos 2 das hiper-heurísticas mais importantes criadas por outros autores ([57], Seção 5.3.2, e [34]). Para estas últimas, experimentamos ainda algumas extensões e modificações. Os dois métodos hiper-heurísticos selecionados podem ser vistos como respectivos representantes de duas classes distintas, que aparentemente englobam todas as hiper-heurísticas já desenvolvidas e nos permitem denominar cada um desses métodos como *hiper-heurística de busca direta por entornos* ou como *hiper-heurística evolutiva indireta*. Implementamos cada hiper-heurística como uma biblioteca (em linguagem C), de forma a evidenciar e estimular a independência entre o nível em que se encontra a hiper-heurística e aquele onde se apresentam as estruturas e abstrações diretamente relacionadas ao problema considerado. Naturalmente, essa separação é de ingente importância para possibilitar a reutilização imediata das hiper-heurísticas e garantir que nelas haja total ausência de informações relativas a um problema de otimização específico.

Abstract

A hyperheuristic [53] is a heuristic that can be used to handle any optimization problem, provided that the algorithm is fed with some parameters, as structures and abstractions, related to the problem at hand. Hyperheuristics have been applied to some practical problems and presented [57, 2, 9, 14, 16, 17, 15, 8, 13, 34, 32, 33, 59] as methods with great potential to allow the quick development of algorithms that are able to successfully deal, from a practical standpoint, with complex ill-known optimization problems. However, it's difficult to position hyperheuristics at some quality level and evaluate their robustness without applying them to problems for which there are many instances available in the public domain and already attacked by worthy algorithms. This work aims to give some important steps towards that process of evaluation, additionally increasing the number of available hyperheuristics, studying the impact of some natural development alternatives and comparing the results obtained by different methods, what also enables us to confront the two classes of hyperheuristics that we have identified. With those purposes in mind, we have developed 3 original hyperheuristics and implemented 2 of the most important hyperheuristics created by other authors ([57] – Section 5.3.2 – and [34]). For those latter two approaches, we have also experimented with some modifications and extensions. The two methods we have chosen for implementation may be seen as respectively representing two distinct classes, which seem to contain all hyperheuristics developed so far and that allow us to classify any of these methods as either being a *direct neighbourhood search hyperheuristic* or an *indirect evolutive hyperheuristic*. We have implemented each hyperheuristic as a library (in the C language), so as to clearly show and estimulate the independence between the level where the hyperheuristic is and that to which the structures and abstractions directly related to the problem at hand belong. Obviously, this separation of concerns is extremely important to make the immediate reuse of hyperheuristics possible and enforce in them the complete absence of information from a specific optimization problem.

Sumário

1 Introdução

O termo “heurística” tem sua origem no verbo grego *heuriskein*, que pode ser traduzido como “encontrar” ou “descobrir”. No contexto da Ciência da Computação, essa palavra não possui um único significado, mas quase sempre está relacionada à ausência de garantias no que diz respeito à utilização de recursos (em especial o tempo) ou à eficácia de um método algorítmico – ou de certas características de um método, como procedimentos e decisões. Neste texto, utilizaremos o adjetivo “heurístico” e o substantivo “heurística” conforme o costume geral no campo da Otimização: um **método heurístico**, ou uma **heurística**, é um algoritmo que objetiva encontrar soluções de qualidade para problemas de otimização, sem oferecer, no entanto, um grau pré-determinado de certeza quanto ao seu consumo de recursos e à qualidade de suas soluções em relação às soluções ótimas (note que não utilizaremos a palavra “solução” como referência restrita à otimalidade ou à viabilidade). Para melhor contextualização, recomendam-se textos sobre complexidade computacional e temas relacionados [50, 21].

A importância dos métodos heurísticos está no fato de que eles são desenvolvidos para lidar com problemas de otimização complexos e representam a única alternativa nos casos em que não existem (ou ainda não foram descobertas) estratégias que proporcionem algum tipo de garantia, como os algoritmos de aproximação ([50], Cap. 17). Além disso, pode ser interessante utilizar uma heurística na tentativa de encontrar soluções superiores às obtidas por um algoritmo que fornece garantias (especialmente se o tempo consumido por ele é muito inferior ao disponível). Em outras situações, é possível que seja necessário um algoritmo extremamente eficiente, ao ponto de ser inevitável que ele seja heurístico, mesmo que o problema não seja complexo. Os métodos heurísticos têm sido aplicados a grande parte dos problemas que surgem na prática, incluindo roteamento de veículos, projeto de redes, escalonamento de tarefas, alinhamento de seqüências de DNA, projeto de circuitos e muitos outros.

Desde o surgimento das heurísticas, diversos métodos importantes foram desenvolvidos para o tratamento de problemas específicos, como o *Neighbor-Joining*, para a construção de árvores filogenéticas a partir de uma matriz de distâncias [56], e as estratégias do sistema *Blackbox*, para a resolução de problemas de planejamento [38]. As heurísticas específicas freqüentemente pos-

suem alta qualidade, mas, na quase totalidade dos casos, não são flexíveis o suficiente para contribuírem de maneira significativa na resolução dos problemas de otimização em geral. Porém, o amplo estudo das características mais influentes no êxito das heurísticas e a necessidade de desenvolvimento de métodos capazes de lidar de forma robusta com a profusão de problemas que surgem no mundo real levaram à elaboração de estratégias chamadas *meta-heurísticas*, que podem ser definidas da seguinte forma:

> "Uma **meta-heurística** é um conjunto de conceitos que pode ser utilizado para definir métodos heurísticos aplicáveis a uma ampla gama de problemas diversos. Em outras palavras, uma meta-heurística pode ser vista como uma estrutura algorítmica geral que pode ser empregada na resolução de diferentes problemas de otimização, com um número relativamente reduzido de modificações que a adaptem para o tratamento de cada problema específico. Dentre as meta-heurísticas existentes, pode-se citar o recozimento simulado (*simulated annealing*), a busca tabu, a busca local iterada (*iterated local search*), os algoritmos evolutivos e a otimização com formigas artificiais (*ant colony optimization*)" ([61], tradução nossa).

Há cerca de três décadas, o desenvolvimento e o estudo das meta-heurísticas vem aprofundando de maneira marcante o conhecimento geral sobre o processo de resolução de problemas complexos. Além disso, as meta-heurísticas são pontos de partida imediatos para a resolução de novos problemas, sem eliminar a possibilidade de incorporação de conhecimentos específicos para a produção de algoritmos mais sofisticados. Dentre as muitas aplicações de meta-heurísticas que obtiveram resultados excelentes, pode-se citar o algoritmo de Lim et al. para a resolução do problema da minimização da largura de banda [42] e a estratégia de Kostuch para lidar com um problema de criação de grades horárias acadêmicas [40].

Apesar do número acentuado de implementações bem-sucedidas de meta-heurísticas, esses métodos, na maioria dos casos, oferecem resultados pobres – do ponto de vista prático – quando não são aperfeiçoados através do emprego de conhecimentos específicos para o problema que se deseja resolver, ou seja, quando são implementados da maneira mais óbvia. Com isso, grande parte dos usuários que dispõem de poucos recursos (especialmente aqueles relacionados ao tempo de desenvolvimento) ainda optam pela utilização de heurísticas simples. Essa dificuldade tem motivado diversos pesquisadores a buscarem métodos ainda mais genéricos que as atuais meta-heurísticas. As *hiper-heurísticas* surgiram como uma conseqüência dessa motivação.

Uma **hiper-heurística**, termo cunhado por Cowling et al. em 2000 [14], é uma heurística que lida indiretamente com o problema que se deseja resolver, através do gerenciamento de heurísticas específicas. Embora essa definição seja bastante ampla, normalmente se fala em

hiper-heurísticas quando se deseja fazer referência a métodos que não utilizam explicitamente informações específicas sobre os problemas explorados. Nesse sentido, uma hiper-heurística é um algoritmo que pode ser adaptado a qualquer problema de otimização, bastando para isso parametrizá-la com informações simples, como uma solução inicial, algumas heurísticas que possam ser implementadas rapidamente e uma função capaz de avaliar as soluções.

A importância das características que formam o conceito de hiper-heurística é evidente, uma vez que, com elas, tais algoritmos têm o potencial de possibilitar a obtenção de soluções de qualidade – comparáveis às que seriam encontradas por implementações cuidadosas de meta-heurísticas – para diversos problemas de otimização, com um esforço de desenvolvimento extremamente reduzido – apenas o necessário para a implementação de heurísticas simples. Naturalmente, uma hiper-heurística também pode se tornar um método específico, caso seja utilizada como primeiro protótipo para o estudo intenso de um único problema. Porém, esse tipo de aplicação não será tratado neste texto, cujo tema focaliza o desenvolvimento de estratégias genéricas.

Como veremos no Capítulo 3, as hiper-heurísticas têm sido empregadas para lidar com alguns problemas práticos. No entanto, é difícil avaliar a robustez dessas abordagens e situá-las em algum nível de qualidade, caso não as apliquemos a problemas para os quais existam diversas instâncias disponíveis publicamente e já experimentadas por algoritmos relevantes. Este trabalho procura dar alguns passos importantes rumo a essas avaliações, além de ampliar o conjunto das hiper-heurísticas, compreender o impacto de algumas alternativas naturais de desenvolvimento e estabelecer comparações entre os resultados obtidos por diferentes métodos, o que ainda nos permite confrontar as duas diferentes classes de hiper-heurísticas (Capítulo 4) que identificamos. Com essas finalidades em mente, desenvolvemos 3 novas hiper-heurísticas e implementamos 2 das hiper-heurísticas mais importantes criadas por outros autores ([57], Seção 5.3.2, e [34]). Para estas últimas, experimentamos ainda algumas extensões e modificações.

A seguir, apresentamos uma visão geral das meta-heurísticas (Capítulo 2) e das hiper-heurísticas (Capítulo 3), enfatizando os métodos mais relevantes para o presente trabalho. Em seguida (Capítulo 4), colocamos as principais motivações para as investigações realizadas e descrevemos as implementações centrais, ou seja, o desenvolvimento das hiper-heurísticas em suas diferentes versões. O Capítulo 5 introduz os dois problemas adotados para experimentação dos algoritmos e descreve as implementações específicas realizadas, discutindo algumas das escolhas que fizemos. O Capítulo 6 explicita os procedimentos utilizados na investigação das hiper-heurísticas e apresenta os mais importantes resultados obtidos com os experimentos. Algumas sugestões para trabalhos futuros aparecem no capítulo final.

2 *Visão Geral das Meta-Heurísticas*

Em 2003, Melián et al. produziram um excelente artigo introdutório sobre meta-heurísticas [45], onde aparecem todos os métodos aqui citados. Sugerimos esse artigo como leitura complementar, embora não requerida.

Não obstante muitas meta-heurísticas diferirem significativamente entre si, três características freqüentemente requerem atenção especial: as soluções inviáveis, a função objetivo e o critério de parada. Cada uma dessas questões pode ser tratada conforme descrito a seguir:

- O critério para lidar com as soluções inviáveis dificilmente é estipulado pelas meta-heurísticas. Portanto, é tarefa do programador solucionar essa questão. A prática mais comum é a *relaxação* do problema, de maneira a tornar viáveis todas as soluções e fazer com que a função objetivo penalize intensamente as soluções outrora inviáveis. Porém, em diversas situações, pode ser interessante adotar alguma outra estratégia, como, por exemplo, impedir a manipulação de soluções inviáveis.

- A função objetivo é um problema relevante apenas nos casos em que avaliar uma solução é uma tarefa computacionalmente custosa. Nessas situações, provenientes da modelagem do problema, normalmente se utiliza uma aproximação simplificada da função objetivo ou se estabelece uma estratégia para avaliar cada nova solução a partir, se for o caso, de uma ou mais soluções que tenham dado origem a ela.

- O critério de parada – estratégia que define o momento em que o algoritmo está concluído – pode ser definido com base em diversos fatores, especialmente nos casos – que constituem a grande maioria – em que a meta-heurística não o define. Entre esses fatores, estão o tempo consumido, o número de iterações realizadas, o número de iterações subseqüentes que não geraram uma solução superior à melhor solução já produzida etc.

Muitos aspectos podem ser considerados ao realizarmos classificações dos métodos meta-heurísticos. Porém, observando as características mais comumente analisadas, é possível notar a existência de duas grandes classes, que enquadram quase todas as meta-heurísticas. Esse fato,

aliado à inexistência de uma categorização padronizada, nos permite dividir esses métodos em **meta-heurísticas de busca por entornos** e **meta-heurísticas populacionais**. Adiante, faz-se uma breve descrição de cada um desses conjuntos de meta-heurísticas, bem como dos métodos que os compõem.

2.1 Meta-Heurísticas de Busca por Entornos

Um *operador*, no contexto dos métodos de busca por entornos, pode ser definido como qualquer procedimento que modifica uma solução parcial ou completa – viável ou não – produzindo outra solução. O termo *vizinhança*, quando relacionado a uma solução *S*, refere-se ao conjunto das soluções que podem ser geradas através da aplicação de algum operador a *S*. Chamamos meta-heurística de busca por entornos a qualquer método que percorra espaços de busca – compostos por soluções – levando em conta fundamentalmente, em cada passo, a vizinhança da solução obtida na iteração anterior. Naturalmente, esses algoritmos requerem uma solução prévia. Essas *soluções iniciais* podem ser produzidas de diversas maneiras: aleatoriamente (por exemplo: uma partição aleatória dos elementos, em um problema de particionamento), trivialmente (como uma solução em que todos os veículos permanecem em suas posições iniciais, em um problema de roteamento), através de métodos construtivos (heurísticas que constróem uma solução de maneira incremental, a partir da seleção metódica de cada componente) ou utilizando quaisquer outras estratégias.

Para a definição e a utilização da *estrutura de entornos* (quantidade de operadores e características destes), questão raramente abordada pela própria meta-heurística, há duas situações recorrentes a serem evitadas: a aplicação e avaliação de um número demasiadamente elevado de operadores em cada iteração, desacelerando a evolução das soluções; a restrição a um pequeno grupo definitivo de operadores aplicáveis, limitando excessivamente o espaço de busca. Com isso, a implementação adequada da estrutura de entornos freqüentemente é conseqüência da experiência do programador e de avaliações experimentais, podendo a carência em um desses fatores ser compensada pela alta disponibilidade do outro.

Em sua concepção primária, as buscas por entornos são *monótonas*, ou seja, nenhuma de suas iterações tem como resultado a atualização da solução corrente de maneira que a nova solução não tenha custo estritamente menor que o da anterior. Essa característica está sintetizada como o clássico método de ***Hill-Climbing*** ([55], Cap. 4), cujo critério de parada consiste em finalizar o algoritmo quando se encontra uma solução S_f cuja vizinhança não oferece possibilidades de aumento imediato de qualidade. Nessa situação, dizemos que foi encontrado um *ótimo local*: a solução S_f. Chamamos de *busca local* qualquer procedimento que tenha como objetivo

encontrar ótimos locais.

Um caso particular do método de *hill-climbing* são os *algoritmos gulosos*, que, em cada iteração, selecionam a melhor solução da vizinhança da solução corrente. Esses algoritmos oferecem, na prática, resultados relevantemente superiores às versões mais simples da busca local, além de garantirem, em alguns problemas e com determinados operadores, a otimização global – Cormen et al. analisaram diversos exemplos desse comportamento ([12], Cap. 16). Porém, muitas vezes a estrutura de entornos não permite a análise completa da vizinhança em cada iteração. Nesses casos, ainda é aconselhável selecionar, em cada passo, uma "boa" solução da vizinhança corrente. Um interessante exemplo relacionado a essa filosofia é a técnica de **intensificação oscilante dinâmica** [45], que altera dinamicamente a *intensidade* da busca (quantidade de soluções analisadas na vizinhança da solução corrente), de forma a aumentar a eficiência do processo nas fases em que há diversas soluções oferecendo diminuição de custo e a torná-lo mais minucioso quando há escassez de soluções com essa característica – o resultado prático é, com freqüência, uma busca com iterações eficientes nas primeiras fases e análise quase completa de vizinhança nos últimos passos em direção ao ótimo local.

O maior inconveniente das meta-heurísticas de busca monótona é a fraca exploração do espaço de busca, uma vez que, em geral, apenas uma ínfima parcela do conjunto de todas as soluções pode ser atingida de maneira monótona a partir da solução inicial. Uma abordagem comum para o tratamento desse problema é a realização de diversas buscas independentes, seguidas pela seleção da melhor solução encontrada. Porém, as meta-heurísticas de busca por entornos que atualmente oferecem os melhores resultados práticos empregam técnicas mais sofisticadas, como veremos a seguir.

A meta-heurística de **Busca por Entornos Variáveis** (*Variable Neighbourhood Search* [47, 35]) emprega um conjunto de estruturas de entornos para realizar alterações sistemáticas na definição de vizinhança. Mesmo em sua forma mais simples, essa meta-heurística possui diferentes versões. A *Busca Gulosa por Entornos Variáveis* (*Variable Neighbourhood Descent*), por exemplo, emprega uma diferente estrutura de entornos em cada iteração, selecionando sempre a melhor solução da vizinhança corrente. O método retorna à primeira estrutura sempre que uma iteração produz diminuição de custo e avança para a próxima estrutura sempre que isso não ocorre – caso não haja mais estruturas, considera-se a execução concluída. A *BEV Reduzida* (*Reduced VNS*) difere da anterior por selecionar uma solução aleatória – e não a melhor – da vizinhança em cada iteração, privilegiando a eficiência. Nesse caso, o processo como um todo pode ser repetido diversas vezes, de acordo com alguma condição de parada. A *BEV Básica* (*Basic VNS*) e a *BEV Geral* (*General VNS*) combinam características das duas variantes anteriores.

A meta-heurística de **Busca Local Guiada** (*Guided Local Search* [60]) é um método de alta

qualidade cuja principal estratégia é penalizar, através de alterações na função objetivo, os elementos (de soluções) que aparecem com freqüência em ótimos locais. Dessa forma, incentiva-se a exploração de diversas regiões do espaço de busca. Deve-se atribuir um custo – em geral estático e baseado na função objetivo – e um fator de penalização – dinâmico e inicializado com zero – a cada possível componente de soluções. Quando a busca atinge um ótimo local, selecionam-se alguns componentes da solução corrente para terem seus fatores de penalização incrementados. Nessa situação, a chance de que um determinado componente seja penalizado é inversamente proporcional ao seu fator de penalização atual e diretamente proporcional ao seu custo. A função objetivo modificada depende da função objetivo original e dos fatores de penalização, aumentando o custo das soluções que contêm elementos penalizados.

A meta-heurística de **Recozimento Simulado** (*Simulated Annealing* [39, 61]) é um método clássico, amplamente estudado e utilizado, capaz de obter resultados de grande qualidade quando executado por um período suficientemente longo. O termo "recozimento" refere-se a um processo físico utilizado na metalurgia para reduzir os defeitos de determinados materiais. Basicamente, o processo consiste em aquecer um material e, em seguida, resfriá-lo lentamente. O recozimento simulado, por sua vez, divide-se em iterações do seguinte procedimento: sorteia-se uma solução S_s da vizinhança da solução corrente (S_c); substitui-se S_c por S_s com probabilidade P (chamada *probabilidade de transição*). Tradicionalmente, estipula-se $P = e^{E/T}$, sendo $E = min\{0, f(S_c) - f(S_s)\}$, f a função de custos e T um número positivo dinamicamente ajustável. Em geral, o valor de T se aproxima lentamente de zero, de forma que a fase final da busca seja semelhante a um algoritmo de *hill-climbing*. Uma versão simplificada do recozimento simulado, o ***Threshold Accepting*** [19], atualiza a solução corrente apenas nas situações em que a inequação $f(S_s) - f(S_c) \leq t$ é satisfeita, sendo t um limitante dinâmico.

A **Busca Tabu** [23, 27] é uma meta-heurística de busca por entornos muito popular, cuja principal característica é a capacidade de exploração do histórico do processo de busca, organizado em estruturas que compõem o que se chama de *memória adaptativa*. A construção e a utilização dessa memória levam em conta quatro *dimensões*, que se referem a certas características das soluções (ou de seus componentes) em relação ao processo de busca: a freqüência, a presença no passado recente, a qualidade e a influência. Por exemplo, podem ser mantidas tabelas que contenham: os elementos que aparecem com maior freqüência nas soluções já encontradas, os elementos mais freqüentes em *soluções críticas* e os elementos que foram adicionados à solução corrente ou removidos dela no passado recente. O conceito de "solução crítica" refere-se às soluções que possuem propriedades especiais, como, por exemplo, a superioridade em relação à solução anterior e à seguinte no processo de busca. A última tabela citada acima – chamada *lista tabu*, uma das estruturas mais importantes da busca tabu – é empregada na classificação

de elementos como *tabu*, o que indica que estes últimos não poderão ser adicionados à solução, ou removidos dela, durante um certo número de iterações. No que diz respeito à qualidade, os *critérios de aspiração* são as estratégias mais utilizadas e têm como função possibilitar a aceitação de soluções de alta qualidade, mesmo nas situações em que haja impedimentos, como elementos presentes na lista tabu.

Dois conceitos muito importantes para a busca tabu – e que têm aparecido com crescente freqüência em outros métodos heurísticos – são a *intensificação* e a *diversificação*. O primeiro conceito refere-se à exploração do espaço de busca através do emprego de elementos de soluções e combinações de movimentos que, de acordo com o histórico da busca, tenham impacto positivo. A diversificação, por sua vez, é a construção de soluções que pertençam a regiões ainda não exploradas do espaço de busca e difiram significativamente das soluções já encontradas.

A **Busca Reativa** [5, 4] é uma extensão da busca tabu, especialmente no que diz respeito à diversificação. Utilizam-se estruturas de dados especiais, para explorar a memória de longo prazo através da verificação explícita da existência de ciclos no processo de busca. Estes últimos são combatidos através de elevações no tamanho da lista tabu ou, em casos extremos, da execução de seqüências aleatórias de movimentos.

A meta-heurística **GRASP** (*Greedy Randomized Adaptive Search Procedures* [52]) é um método formado pela reiteração de um processo de duas fases: construção gradual inteligente e busca por entornos operando sobre soluções completas. A fase construtiva parte de uma solução parcial vazia e, em cada passo: identifica todos os elementos que podem ser incorporados à solução parcial sem torná-la inviável; cria um subconjunto desses elementos, contendo apenas aqueles que causam aumento mínimo de custo (este é o aspecto guloso do método); seleciona aleatoriamente um elemento desse subconjunto, acrescentando-o à solução parcial corrente. A segunda fase de cada iteração do método GRASP é, em geral, bastante simples, consistindo apenas na execução de um algoritmo de *hill-climbing*.

A meta-heurística de **Busca Local Iterada** (*Iterated Local Search* [44]) opera uma heurística de busca por entornos H, definida no momento da implementação. O único comportamento que se espera da heurística H é que, dada uma solução, ela produza um ótimo local, de acordo com seus próprios critérios – não necessariamente conhecidos pela estrutura principal do método de BLI. Mais especificamente, cada iteração da meta-heurística deve: efetuar alterações na solução corrente, através da observação do histórico do processo, de forma a produzir uma nova solução S_n (este passo, naturalmente, é dispensável na primeira iteração); aplicar H à solução S_n, gerando um ótimo local S_o; fazer de S_o a nova solução corrente, caso ela satisfaça determinados critérios baseados em S_n, no histórico da busca e na solução corrente. O ideal dessa meta-heurística é a realização de uma busca por entornos indireta no conjunto dos ótimos

locais definidos por *H*.

2.2 Meta-Heurísticas Populacionais

As meta-heurísticas populacionais lidam com um conjunto de soluções, muitas vezes chamado de *população*, que evolui, através da interação entre seus elementos e de processos individuais, procurando enfatizar as características desejáveis das soluções em toda a população, num processo que objetiva aumentar a qualidade média sem comprometer a diversidade do conjunto. Alguns exemplos de meta-heurísticas evolutivas são os **Algoritmos Genéticos** (Seção 2.3), os **Algoritmos Meméticos** (Seção 2.3), os **Algoritmos de Estimação de Distribuição** (Seção 2.5), a **Busca Dispersa** (Seção 2.4), a **Reconexão de Caminhos** (Seção 2.4), a **Otimização com Formigas Artificiais** (*Ant Colony Optimization* [18]) e o método ***Particle Swarm Optimization*** [37, 51].

A otimização com formigas artificiais baseia-se no comportamento utilizado pelas colônias de formigas para traçar rotas entre o formigueiro e as fontes de alimentação. O principal aspecto desse comportamento é a secreção de uma substância chamada *feromônio*, através da qual uma formiga indica às demais alguns caminhos que a levaram aos alimentos. A meta-heurística emprega formigas artificiais, representadas por processos concorrentes, que traçam caminhos em um grafo cujas arestas representam componentes de soluções.

A meta-heurística *particle swarm optimization* foi desenvolvida pelo psicólogo James Kennedy e pelo engenheiro Russell Eberhart, com inspiração nos modelos do biólogo Frank Heppner para o comportamento social dos pássaros em revoadas. A meta-heurística lida com um conjunto de *partículas* e possui as seguintes características:

1. Cada partícula armazena uma única solução em cada instante. Essa solução pode ser vista como a posição da partícula no espaço de busca.

2. Cada partícula possui uma velocidade dinâmica. O conceito exato de velocidade é definido no momento da implementação, embora a escolha mais simples seja a utilização de vetores euclidianos de números reais.

3. Em cada iteração, as velocidades das partículas são atualizadas e cada partícula se move – ou seja, altera a sua solução – de acordo com sua nova velocidade.

4. Para redefinir sua velocidade, cada partícula leva em conta, além de sua velocidade corrente, a melhor solução que ela já conteve e a melhor solução que os seus *vizinhos* (item 5) já encontraram. O peso dado a cada uma dessas influências é escolhido aleatoriamente

por cada partícula, em cada iteração. Dessa forma, define-se aleatoriamente a *individualidade* e a *sociabilidade* de cada partícula. Cabe notar que os pesos são independentes, o que torna possível, por exemplo, que uma partícula tenha grande individualidade e grande sociabilidade em uma mesma iteração – embora seja possível estabelecer limites de velocidade.

5. As partículas compõem uma topologia ou, mais formalmente, um grafo não dirigido em que cada partícula é um vértice e cada aresta representa uma relação de vizinhança. É importante notar que o conceito de vizinhança, nesta meta-heurística, não está relacionado ao espaço de busca.

6. A topologia deve ser definida de forma que cada partícula possa influenciar, ainda que indiretamente, todas as demais. Em outras palavras, o grafo que representa a topologia deve ser conexo.

2.3 Introdução aos Algoritmos Genéticos e Meméticos

Os **algoritmos genéticos** [6] são os métodos mais populares dentre as meta-heurísticas evolutivas e já foram aplicados a um grande número de problemas de otimização, nos mais diversos contextos. Esses algoritmos se inspiram nos processos de seleção natural que ocorrem em populações de seres vivos, característica que trouxe termos como “geração”, “gene” e “aptidão” para o contexto dos métodos heurísticos. Os **algoritmos meméticos** [48] são extensões dos algoritmos genéticos e sua principal particularidade é o emprego de otimização local.

2.3.1 Algoritmos Genéticos

A forma mais simples de representar os algoritmos genéticos está ilustrada na Figura 2.1. A seguir, definimos os principais elementos desses métodos.

Nos algoritmos genéticos, cada solução deve ser representada como um *cromossomo*, que consiste em uma seqüência de símbolos, chamados *genes*. O algoritmo deve trabalhar com um conjunto dinâmico de cromossomos – também chamados de *indivíduos* –, denominado *população*. Cada cromossomo possui uma *aptidão*, que é uma medida de qualidade obtida através da *função de aptidão*. Esta última, é claro, deve estar intimamente relacionada à função objetivo.

Na *fase reprodutiva* do algoritmo genético, selecionam-se indivíduos da população para sofrerem *recombinação*, o que dá origem à prole que constitui a geração seguinte. Durante a seleção, os indivíduos mais aptos devem ser favorecidos, de forma que, por exemplo, a proba-

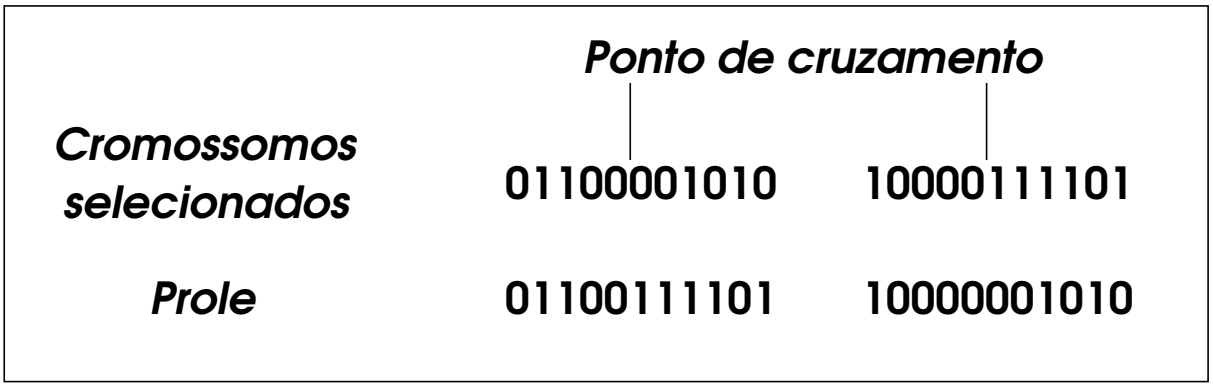
Figura 2.1: Algoritmo Genético Básico.

1. Construa a população inicial, com *TAM_POP* indivíduos.
2. Enquanto o critério de parada não estiver satisfeito:
   (a) Calcule a aptidão de cada indivíduo da população atual.
   (b) Execute o procedimento abaixo $\frac{TAM\_POP}{2}$ vezes, de maneira a produzir a nova geração de *TAM_POP* indivíduos:
      i. Selecione 2 indivíduos da população atual.
      ii. Recombine os 2 indivíduos selecionados, de maneira a produzir 2 novos indivíduos.
      iii. Inclua os 2 novos indivíduos na próxima geração.

bilidade de seleção seja proporcional à aptidão. A recombinação é composta pelas fases de *cruzamento* e *mutação*, nesta ordem.

A forma mais básica de cruzamento entre dois cromossomos consiste no intercâmbio entre os segmentos finais de ambos – o que é conhecido como *cruzamento de ponto único* –, a partir de uma posição aleatoriamente escolhida (confira na Figura 2.2). O cruzamento não é necessariamente aplicado a todos os pares de cromossomos selecionados. Deve-se definir uma *taxa de cruzamento*, que é um número tipicamente no intervalo $[\frac{6}{10}, 1]$ e indica a probabilidade de execução daquele passo. Caso o cruzamento não seja efetuado, a prole pré-mutação será composta pelos próprios indivíduos selecionados. A mutação, quando aplicada a um cromossomo, altera aleatoriamente cada um de seus genes com uma probabilidade baixa (por exemplo: $0,001$).

Figura 2.2: Cruzamento de Ponto Único entre Par de Cromossomos.



Como observa Goldberg ([28] apud Beasley et al. [6]), o poder dos algoritmos genéticos está na capacidade de explorar os "blocos de construção" (*building blocks*) de qualidade encontrados

durante o processo de busca. Esses blocos podem ser definidos como pequenas seqüências de genes que têm boa influência na qualidade da solução. Portanto, a codificação das soluções deve ser realizada de forma a se aproximar o máximo possível dos seguintes ideais:

- Os genes mais fortemente interrelacionados devem estar em posições mais próximas no cromossomo.
- Deve haver pouca *epistasia*, ou *interação*, entre os genes, ou seja, a qualidade de um gene deve estar condicionada o mínimo possível aos valores de outras posições do cromossomo.

A passagem entre gerações (ou seja: a substituição de população) no algoritmo genético é tipicamente efetuada com o auxílio de uma *população intermediária* (*mating pool*). Essa população é composta por indivíduos selecionados da população corrente e possui o mesmo tamanho desta (naturalmente, a população intermediária, em geral, contém cromossomos repetidos). Com isso, a população seguinte pode ser produzida através de um processo que, em cada iteração, remove dois indivíduos (selecionados aleatoriamente) da população intermediária e os recombina, de forma a gerar dois elementos para a próxima população.

A construção da população intermediária é normalmente efetuada de forma que o número de cópias de cada indivíduo presentes nessa população fique próximo da razão entre a aptidão do indivíduo e a aptidão média na população corrente. Esse efeito pode ser obtido através do *remapeamento explícito de aptidão* (a aptidão de cada indivíduo é diretamente transformada, através de uma função, em um número que indica o número de cópias presentes na população intermediária) ou com o *remapeamento implícito de aptidão* (o número de cópias de cada cromossomo não é calculado antes da construção da população intermediária, mas os indivíduos mais aptos são selecionados com maior probabilidade).

O *remapeamento explícito de aptidão* freqüentemente sofre de males relacionados ao tempo de convergência do algoritmo, ou seja, ao tempo decorrido até que, para cada posição i, exista um símbolo s_i que ocorra nessa posição em 95% dos cromossomos da população. Mais informalmente, diz-se que um algoritmo genético convergiu quando quase todos os cromossomos da população se tornam “muito parecidos” e a aptidão média na população se aproxima muito da aptidão de seu melhor indivíduo. Para evitar a convergência precoce, ou excessivamente demorada, de um algoritmo genético, é comum realizar um mapeamento intermediário das aptidões, normalmente para comprimir ou expandir a faixa de valores em que aparecem as razões entre as aptidões dos indivíduos e a aptidão média.

Uma maneira alternativa de realizar a passagem entre gerações ocorre no *algoritmo genético de*

estado regular (*steady-state genetic algorithm*), que substitui apenas dois indivíduos da população em cada passagem. Mais especificamente, a população é atualizada através de um processo que simplesmente seleciona dois indivíduos e os recombina, produzindo dois novos cromossomos, que serão utilizados para substituir duas soluções da população corrente. Com isso, a passagem entre gerações envolve, no total, a seleção de quatro indivíduos. Cada um desses cromossomos pode ser selecionado aleatoriamente ou com base na aptidão, desde que a aleatoriedade não seja empregada nos quatro casos. Embora o algoritmo genético de estado regular freqüentemente obtenha melhor aptidão média populacional nos instantes iniciais da busca, não há evidência de superioridade desse método perante os demais algoritmos genéticos.

2.3.2 Algoritmos Meméticos

Os algoritmos meméticos são métodos heurísticos que combinam características de meta-heurísticas evolutivas com processos de busca por entornos. Embora esses métodos se assemelhem fortemente aos algoritmos genéticos, suas particularidades permitem classificá-los como constituintes de uma meta-heurística distinta. A seguir, apresentamos brevemente os algoritmos meméticos (para uma introdução mais completa, consulte Moscato e Cotta-Porras [48]).

Um algoritmo memético deve trabalhar com uma população de *agentes*, termo que indica a maior autonomia desses elementos, quando comparados aos cromossomos dos algoritmos genéticos. Cada agente deve conter ao menos uma solução em cada instante e deve ser avaliado conforme uma função que se baseie nos custos das soluções que ele armazena. O esquema geral dos algoritmos meméticos está representado na Figura 2.3. Observe que esses algoritmos podem utilizar diversos operadores, embora seja mais comum o emprego de cruzamento, mutação e otimização local, seguindo estratégias herdadas dos algoritmos genéticos. Variações possíveis são as mutações mais sofisticadas (com procedimentos específicos para o problema em foco) e o cruzamento entre três ou mais indivíduos.

Uma forma alternativa de enxergar a população de um algoritmo memético é como um conjunto de agentes que realizam explorações autônomas do espaço de busca e periodicamente cooperam entre si. A população inicial pode ser produzida de diversas maneiras, embora as estratégias mais comuns se baseiem em aleatoriedade, otimização local e métodos heurísticos mais sofisticados. Por exemplo, a população inicial poderia ser construída com um conjunto de soluções obtidas durante a execução de um algoritmo baseado em recozimento simulado. A convergência da população pode ser determinada, por exemplo, utilizando-se a fórmula da entropia de Shannon (consulte a Seção 2.5).

Figura 2.3: Esquema Geral dos Algoritmos Meméticos.

1. Faça $PopAtual = \emptyset$.
2. Repita TAM_POP vezes:
   - Construa um agente A_a de alguma forma (por exemplo, aleatoriamente) e realize otimização local a partir dele, para obter um novo agente A_o.
   - Faça $PopAtual = PopAtual \cup \{A_o\}$.
3. Seja op o vetor que contém os operadores a serem utilizados na fase de recombinação (este vetor deve conter ao menos um otimizador local).
4. Enquanto o critério de parada não estiver satisfeito:
   (a) Utilize um critério de seleção para construir a população R_0, com alguns elementos de $PopAtual$.
   (b) Para $i = 1, 2, \ldots, |op|$:
      - Construa a população R_i a partir de R_{i-1}, com o emprego do operador $op[i]$.
   (c) Atualize a população atual ($PopAtual$), confrontando-a com a população $R_{|op|}$.
   (d) Se a população atual convergiu, reinicialize-a desta forma:
      i. Estipule o valor de $cons$, indicando o número de elementos que serão conservados.
      ii. Elimine de $PopAtual$ seus $(TAM\_POP - cons)$ piores agentes.
      iii. Acrescente $(TAM\_POP - cons)$ agentes a $PopAtual$, utilizando, por exemplo, o mesmo método empregado na criação da população inicial.

2.4 Introdução à Busca Dispersa e à Reconexão de Caminhos

A **reconexão de caminhos** [22, 26] e a **busca dispersa** [22, 24, 26] são métodos populacionais capazes de lidar de forma desenvolta com diversos problemas de otimização, o que pode ser ilustrado com a análise de exemplos realizada por Glover et al. [25]. Os principais diferenciais dessas meta-heurísticas em relação a outros métodos evolutivos são o tamanho da população (necessariamente, utilizam-se poucas soluções) e a restrição ao emprego de aleatoriedade, de acordo com os princípios da busca tabu. A filosofia da reconexão de caminhos e da busca dispersa pode ser sintetizada por estas observações e estratégias:

- Lidando-se com uma coleção de soluções bastante distintas e de qualidade, que se pode chamar de *conjunto de referência*, freqüentemente se obtêm informações úteis para a construção de soluções ótimas.

- A construção de combinações entre soluções, bem como a aplicação de processos heurísticos para a melhoria das soluções produzidas com essa estratégia, contribui significativamente para a qualidade de um método.
- O processo de combinação de soluções não deve se limitar à região do espaço de busca delimitada pelo conjunto de referência.
- O emprego de combinações entre mais de duas soluções é capaz de elevar a qualidade de uma meta-heurística evolutiva, devido à maior exploração das soluções de baixo custo já obtidas.

2.4.1 Busca Dispersa

A forma básica da busca dispersa é composta por cinco procedimentos:

- O *método de geração de diversificação* produz uma coleção de soluções diversas, possivelmente a partir de uma ou mais soluções.
- O *método de melhora* recebe uma solução S e produz uma ou mais soluções que não tenham custo superior ao de S.
- O *método de geração de subconjuntos* produz subconjuntos do conjunto de referência, que serão úteis para a construção de combinações de soluções. Por exemplo, este método pode simplesmente produzir todos os pares de soluções do conjunto de referência.
- O *método de combinação de soluções* transforma um conjunto C de soluções em uma ou mais combinações dos elementos de C. Tipicamente, este método realiza combinações lineares (convexas ou não) entre soluções representadas como elementos de um espaço euclidiano.

O algoritmo da Figura 2.4 ilustra uma forma simples, que enfatiza a intensificação, para se aplicar a busca dispersa a um problema de minimização com função objetivo f. É importante ressaltar que:

- A variável b (normalmente com valor não superior a 20) representa o tamanho do conjunto de referência.
- O valor da variável *TamP* é, em geral, consideravelmente maior que o de b (por exemplo, $10 \times b$).

- O resultado do algoritmo (que deve ser a melhor solução viável encontrada durante todo o processo) é, em geral, o elemento x_1 do conjunto de referência final.

Figura 2.4: Uma Versão Simples da Busca Dispersa.

1. Utilizando o método de geração de diversificação, construa um conjunto A, com $TamP$ soluções.
2. Aplique o método de melhora a cada elemento de A, de forma a produzir um novo conjunto P, com soluções de menor custo.
3. Construa o conjunto de referência $CjRef = \{x_1, x_2, \ldots, x_b\}$, contendo b elementos de P.
4. Organize $CjRef$ em ordem crescente de custo, de forma que a melhor solução seja x_1 e a pior seja x_b.
5. Faça $NovasSolucoes = 1$.
6. Enquanto $NovasSolucoes = 1$:
   (a) Produza $NovosSubconjuntos$, contendo todos os pares de soluções $\{\alpha, \beta\}$ tais que $\alpha, \beta \in CjRef$.
   (b) Faça $NovasSolucoes = 0$.
   (c) Enquanto $NovosSubconjuntos \neq \emptyset$:
      i. Extraia um conjunto D de $NovosSubconjuntos$.
      ii. Aplique o método de combinação de soluções a D, obtendo um conjunto G de soluções.
      iii. Aplique o método de melhora a cada elemento de G, de forma a construir um novo conjunto H de soluções de menor custo.
      iv. Para cada elemento s de H:
         - Se $s \notin CjRef$ e $f(s) < f(x_b)$: faça $x_b = s$, reordene $CjRef$ e faça $NovasSolucoes = 1$.

Idealmente, o conjunto de referência inicial é composto por um b-subconjunto de P tal que a soma das distâncias entre suas soluções seja máxima (observe que deve ser definido o conceito de distância entre soluções). Porém, pode-se utilizar, por exemplo, qualquer método que, em cada iteração, insira em $CjRef$ um elemento de P cuja soma das distâncias aos elementos correntes do conjunto de referência seja máxima.

2.4.2 Reconexão de Caminhos

A reconexão de caminhos introduz uma estratégia mais específica para a combinação entre soluções: estas devem ser conectadas através de seqüências compostas por novas soluções. Por exemplo, podem-se combinar duas soluções quaisquer (digamos, x' e x'') através de um processo que parte de x' e, em cada iteração, altera a solução corrente de maneira a aumentar sua semelhança com x''. Dessa forma, produz-se uma seqüência (também chamada de "caminho"), de tamanho r (valor que, naturalmente, não precisa ser conhecido de antemão), $x' = y(1), y(2), \ldots, y(r) = x''$.

Dependendo do número de soluções produzidas em cada seqüência, pode ser interessante aplicar o método de melhora a apenas algumas dessas soluções. Algumas formas de tornar essa estratégia possível são:

- Introduzir uma variável *IntervMelh* e aplicar o método de melhora a cada *IntervMelh* iterações em um mesmo caminho.
- Introduzir uma variável *NumMelh* e aplicar o método de melhora às *NumMelh* melhores soluções produzidas em cada caminho.

O conjunto de referência inicial da reconexão de caminhos pode ser construído da mesma forma empregada pela busca dispersa, embora o procedimento recomendado seja a obtenção de uma coleção de soluções de qualidade a partir da execução de um método de busca por entornos. A Figura 2.5 exibe uma versão simples da reconexão de caminhos, criada a partir do algoritmo de busca dispersa ilustrado anteriormente. Observe que os caminhos produzidos conectam cada par de soluções de duas formas distintas.

2.4.3 Estratégias Avançadas

Algumas técnicas são freqüentemente empregadas para aumentar o potencial dos dois algoritmos aqui apresentados. Dentre essas técnicas, destacamos:

- *Reconstrução do conjunto de referência*: os algoritmos apresentados se consideram concluídos quando o conjunto de referência permanece o mesmo durante uma iteração completa do laço mais externo. Naturalmente, uma possível conseqüência dessa estratégia é a terminação do algoritmo antes que todo o tempo disponibilizado pelo usuário tenha sido explorado. Esse problema pode ser evitado com uma simples reconstrução do conjunto de referência: eliminam-se as $\lceil b/2 \rceil$ piores soluções do conjunto, produz-se outra versão

Figura 2.5: Uma Versão Simples da Reconexão de Caminhos.

1. Construa $CjRef = \{x_1, x_2, \ldots, x_b\}$, contendo b soluções diversas e de qualidade, em ordem crescente de custo.
2. Faça $NovasSolucoes = 1$.
3. Enquanto $NovasSolucoes = 1$:
   (a) Produza $NovosSubconjuntos$, contendo todos os pares de soluções $\{\alpha, \beta\}$ tais que $\alpha, \beta \in CjRef$.
   (b) Faça $NovasSolucoes = 0$.
   (c) Enquanto $NovosSubconjuntos \neq \emptyset$:
      i. Extraia um par $\{x', x''\}$ de $NovosSubconjuntos$.
      ii. Aplique o método de reconexão para produzir uma seqüência $x' = y(1), y(2), \ldots, y(r) = x''$.
      iii. Faça $i = 2$. Enquanto $i < r$:
         - Aplique o método de melhora a $y(i)$.
         - Faça $i = i + IntervMelh$.
      iv. Aplique o método de reconexão para produzir uma seqüência $x'' = w(1), w(2), \ldots, w(t) = x'$.
      v. Faça $i = 2$. Enquanto $i < t$:
         - Aplique o método de melhora a $w(i)$.
         - Faça $i = i + IntervMelh$.
      vi. Para cada solução s produzida:
         - Se $s \notin CjRef$ e $f(s) < f(x_b)$: faça $x_b = s$, reordene $CjRef$ e faça $NovasSolucoes = 1$.

do conjunto P utilizado pela busca dispersa e incluem-se $\lceil b/2 \rceil$ soluções desse conjunto em $CjRef$, com a mesma estratégia anteriormente empregada.

- *Atualização dinâmica do conjunto de referência*: quando a pior solução do conjunto de referência é substituída por outra, esta nova solução será combinada com outras apenas quando estiver concluída a iteração corrente do laço principal. Para evitar esse comportamento, basta igualar $NovosSubconjuntos$ ao conjunto vazio sempre que uma solução do conjunto de referência for substituída. Porém, é importante levar em conta uma conseqüência (não necessariamente negativa) dessa nova estratégia: muitas soluções não terão a oportunidade de se combinarem com outras, mesmo tendo estado presentes no conjunto de referência em algum instante.

- *Conexão e extrapolação*: se o problema a ser resolvido for representado de tal forma que

as soluções sejam compostas por atributos que possam ser removidos ou acrescentados por operadores, a conexão entre duas soluções, x' e x'', e a extrapolação dessa conexão podem ser efetuadas de acordo com o esquema abaixo:

  - Em cada passo, selecione um movimento que acrescente à solução corrente alguns atributos que estejam contidos em x''. Como objetivo secundário, dê preferência aos movimentos que eliminem da solução corrente poucos atributos presentes em x'.
  - Uma vez que x'' tenha sido atingida, o objetivo de cada movimento será acrescentar à solução corrente alguns atributos que não estejam presentes em x'. Como critério secundário, deve-se dar preferência aos movimentos que removam da solução corrente o menor número possível de atributos presentes em x''.

- *Combinações entre mais de duas soluções*: a busca dispersa e a reconexão de caminhos não se limitam às combinações entre pares de soluções. Porém, não é desejável que se produzam todos os subconjuntos do conjunto de referência, uma vez que essa operação deve ser realizada diversas vezes e que cada um desses subconjuntos pode dar origem a muitas outras soluções. Glover [22] propõe a utilização dos seguintes subconjuntos:

  - Todos os pares de soluções.
  - Todo conjunto de 3 elementos que possa ser formado a partir de um par $\{\alpha,\beta\}$ através da adição da melhor solução presente em $CjRef - \{\alpha,\beta\}$.
  - Todo conjunto de 4 elementos que possa ser formado a partir de um conjunto $\{\alpha,\beta,\gamma\}$ do item anterior através da adição da melhor solução presente em $CjRef - \{\alpha,\beta,\gamma\}$.
  - Todo conjunto que contenha os i melhores elementos de $CjRef$ para algum i tal que $i > 4$ e $i \leq b$.

  Para a produção de combinações entre todas as soluções contidas em um determinado conjunto C, pode-se empregar qualquer método que parta de uma dessas soluções, que chamaremos de x', e siga estratégias semelhantes às propostas no item anterior, mas levando em conta, em lugar dos atributos de uma solução x'', todos os atributos das soluções de $C - \{x'\}$, possivelmente estipulando pesos para eles (de acordo com o número de soluções em que cada atributo está presente).

2.5 Introdução aos Algoritmos de Estimação de Distribuição

Os **algoritmos de estimação de distribuição** [41] representam uma meta-heurística evolutiva cuja principal característica é a construção de soluções de forma completamente aleatória, com o emprego de alguma distribuição de probabilidades que evolua durante a execução. A essência dessa meta-heurística está sintetizada na Figura 2.6. Note a existência de um *critério de seleção*, tipicamente baseado na função objetivo.

Figura 2.6: Forma Geral dos Algoritmos de Estimação de Distribuição.

1. Construa um conjunto D_0, com M soluções produzidas aleatoriamente.
2. Faça $l = 1$.
3. Enquanto o critério de parada não estiver satisfeito:
   - Obtenha N indivíduos de D_{l-1}, de acordo com o critério de seleção, e construa D_{l-1}^{Se}, contendo as N soluções escolhidas.
   - Defina a distribuição de probabilidades p_l, de acordo com os indivíduos selecionados: $p_l(x) = p(x|D_{l-1}^{Se})$, para toda solução x.
   - Construa D_l, contendo M indivíduos produzidos, aleatoriamente, de acordo com a distribuição p_l.

Os principais algoritmos de estimação de distribuição consideram que as soluções são representadas como vetores de variáveis discretas, o que os torna mais adequados a problemas de otimização combinatória. Esses algoritmos diferem entre si, principalmente, na abordagem das dependências entre as variáveis aleatórias. A seguir, descrevemos sucintamente algumas abordagens comuns.

O **UMDA** (*Univariate Marginal Distribution Algorithm* [49]), criado por Mühlenbein em 1998, é um algoritmo de estimação de distribuição que, supondo independência entre as variáveis, atualiza a distribuição de probabilidades, em cada iteração, através da contagem de freqüência de cada elemento nas soluções selecionadas. Mais especificamente, o passo de atualização da distribuição é efetuado desta forma:

$$p_l(x) = p(x|D_{l-1}^{Se}) = \prod_{i=1}^{n} p_{li}(x_i) = \prod_{i=1}^{n} \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i|D_{l-1}^{Se})}{N}$$

Sendo:

- $X_1, X_2, \ldots, X_n$ variáveis discretas.

- x um vetor de dimensão n, composto pelos elementos $x_1, x_2, \ldots, x_n$, sendo x_i o valor de X_i em x, para cada $i \in \{1, \ldots, n\}$.
- $\delta_j(X_i = x_i | D_{l-1}^{Se}) = \begin{cases} 1 & \text{se } X_i = x_i \text{ no } j\text{-ésimo indivíduo de } D_{l-1}^{Se} \\ 0 & \text{caso contrário} \end{cases}$

Como ilustração do método UMDA, considere o problema extremamente simples que consiste na maximização da soma dos elementos de um vetor com $n = 4$ e variáveis binárias. Sejam $M = 10$ e $N = 5$. A Tabela 2.1 contém uma possível população inicial D_0, enquanto a Tabela 2.2 contém os indivíduos selecionados na primeira iteração (D_0^{Se}). Dessa forma, a distribuição p_1 seria tal que $p_{11}(1) = \frac{3}{5}$, $p_{12}(1) = \frac{4}{5}$, $p_{13}(1) = \frac{4}{5}$ e $p_{14}(1) = \frac{1}{5}$, de modo que a população D_1 provavelmente seria superior a D_0, como no exemplo da Tabela 2.3.

Tabela 2.1: Uma Possível População Inicial para o UMDA.

Indivíduo	X_1	X_2	X_3	X_4	Soma
1	0	1	1	0	2
2	0	0	0	1	1
3	1	0	0	0	1
4	1	1	1	0	3
5	1	1	0	0	2
6	1	0	1	0	2
7	0	1	0	0	1
8	0	0	1	0	1
9	0	1	1	1	3
10	0	0	0	0	0

Tabela 2.2: Indivíduos da População Inicial Selecionados pelo UMDA.

Indivíduo	X_1	X_2	X_3	X_4	Soma
1	0	1	1	0	2
4	1	1	1	0	3
5	1	1	0	0	2
6	1	0	1	0	2
9	0	1	1	1	3

O algoritmo **PBIL** – *Population Based Incremental Learning* ([3] apud Larrañaga et al. [41]) – é uma forma mais geral do UMDA, uma vez que permite atualizar mais suavemente a distribuição de probabilidades, evitando a desconsideração completa das probabilidades inferidas nas iterações anteriores. Deve ser definida uma distribuição inicial de probabilidades (p_0), bem

Tabela 2.3: População no Fim da Primeira Iteração do UMDA.

Indivíduo	X_1	X_2	X_3	X_4	Soma
1	0	1	1	0	2
2	1	1	1	0	3
3	1	0	1	0	2
4	0	1	1	0	2
5	1	1	1	0	3
6	0	1	1	0	2
7	0	0	1	1	2
8	1	1	1	0	3
9	1	1	0	0	2
10	1	0	1	0	2

como um parâmetro $\alpha \in (0,1]$, que indica o peso da iteração corrente na atualização das probabilidades. Dessa forma, a equação de substituição de probabilidades se modifica ligeiramente:

$$p_l(x) = \prod_{i=1}^{n} p_{li}(x_i) = \prod_{i=1}^{n} (1-\alpha) p_{(l-1)i}(x_i) + \alpha \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_{l-1}^{Se})}{N}$$

Em 1997, De Bonet et al. [7] apresentaram o método **MIMIC** (*Mutual Information Maximization for Input Clustering*), que leva em consideração dependências entre pares de variáveis, diferindo significativamente dos algoritmos UMDA e PBIL. Considere a distribuição de probabilidade conjunta de n variáveis, que, com $X = \{X_1, X_2, \ldots, X_n\}$, pode ser definida como:

$$p(X) = p(X_1|X_2,\ldots,X_n)\, p(X_2|X_3,\ldots,X_n)\, \ldots\, p(X_{n-1}|X_n)\, p(X_n)$$

O objetivo do algoritmo MIMIC, em cada iteração, é encontrar uma distribuição de probabilidades p_l^{π} que esteja o mais próximo possível da distribuição de probabilidade conjunta mais adequada ao conjunto de indivíduos escolhidos pelo critério de seleção e que tenha a seguinte forma:

> $p_l^{\pi}(X) = p(X_{i_1}|X_{i_2})\, \ldots\, p(X_{i_{n-1}}|X_{i_n})\, p(X_{i_n})$, sendo π uma permutação $(i_1, i_2, \ldots, i_n)$ dos índices $1, 2, \ldots, n$.

Encontrar a permutação π^* desejada significa determinar a distribuição, na forma acima (com dependências entre pares de variáveis), que menos difere da distribuição conjunta sugerida pelo conjunto de indivíduos considerados. Utilizando o conceito de divergência de Kullback-Leibler,

ou entropia relativa, entre duas distribuições, chegamos à fórmula:

$$\pi^* = argmin_\pi\{H_l^\pi = h_l(X_{i_n}) + \sum_{j=1}^{n-1} h_l(X_{i_j}|X_{i_{j+1}})\}$$

Onde:

- $h(Z) = -\sum_z p(Z=z)\log_2 p(Z=z)$ é a entropia de Shannon da variável Z.
- $h(Z|Y) = \sum_y h(Z|Y=y)p(Y=y)$ é a entropia (ou incerteza) condicional de Z dado Y.
- $h(Z|Y=y) = -\sum_z p(Z=z|Y=y)\log_2 p(Z=z|Y=y)$ é a entropia condicional de Z dado que $Y=y$.

Nas fórmulas acima, as probabilidades ditadas por p são determinadas a partir de contagens efetuadas em cada um dos indivíduos selecionados. Na busca por π^*, De Bonet et al. utilizam um algoritmo guloso heurístico, que evita a análise de todas as permutações, como ilustrado na Figura 2.7.

Figura 2.7: Busca Gulosa por π^*, Proposta por De Bonet et al.

1. Faça $i_n = argmin_j h_l(X_j)$.
2. Para $k = n-1, n-2, \ldots, 1$:
   - Faça $i_k = argmin_{j \notin \{i_{k+1},\ldots,i_n\}}\ h_l(X_j|X_{i_{k+1}})$.

Outros algoritmos de estimação de distribuição, incluindo alguns métodos que levam em conta as dependências em subconjuntos com mais de duas variáveis, foram brevemente discutidos por Larrañaga et al. [41], que apontam um grande número de referências associadas.

3 Visão Geral das Hiper-Heurísticas

O termo “hiper-heurística” pode se referir a qualquer processo heurístico que administre outras heurísticas na resolução de problemas de otimização. Uma implementação natural de uma hiper-heurística é, por exemplo, um algoritmo que recebe um conjunto de heurísticas e, em cada ponto de decisão, seleciona uma delas para aplicar no passo seguinte. Porém, a principal motivação para o estudo das hiper-heurísticas é a potencial combinação entre facilidade de aplicação e robustez, o que nos leva a enfatizar os métodos cuja inteligência é quase completamente construída sem o emprego de informações sobre um problema específico de otimização. Essa característica pode ser garantida através, por exemplo, da adoção da arquitetura de Ross et al. [53], que, com base em algumas implementações anteriores [14, 15, 16, 34, 17, 13], estipula algumas regras gerais para a construção de hiper-heurísticas.

Em qualquer implementação que esteja de acordo com a arquitetura de Ross et al., a hiper-heurística opera no nível mais alto, sem utilizar diretamente informações sobre o problema cuja resolução é o objetivo final. No outro nível, apresentam-se as heurísticas que lidam explicitamente com o problema considerado, bem como as demais informações e procedimentos específicos para o problema. Tudo que for transmitido do nível mais baixo para a hiper-heurística deve passar por uma interface estática, de forma que o nível mais alto do método jamais requeira alterações, mesmo que o problema se modifique. A hiper-heurística pode exigir que o nível mais baixo disponibilize diversas capacidades de comunicação. Por exemplo, ela pode requerer que as heurísticas de baixo nível, ao serem chamadas para execução, sejam capazes de receber um parâmetro que indique o instante em que a heurística deve fornecer resultados ao nível mais alto.

Naturalmente, a facilidade de aplicação e a qualidade de uma hiper-heurística tendem a estar fortemente ligadas à quantidade e aos tipos de informações transmitidas através da interface com o nível mais baixo. Minimizando e simplificando a interface, teremos um método aplicável de forma trivial a muitos problemas de otimização, mas provavelmente incapaz de oferecer resultados aceitáveis de maneira robusta, ou seja, de atacar diversos problemas com desenvoltura. A grande maioria dos métodos hiper-heurísticos existentes consistem em algoritmos que recebem

como parâmetros:

- Uma solução para a instância de problema a ser tratada.
- Um conjunto de heurísticas simples. Cada uma dessas heurísticas deve ser capaz de receber uma solução e produzir outra a partir dela.
- Uma função capaz de avaliar soluções.

Dessa forma, é fácil perceber que as hiper-heurísticas são algoritmos bem definidos e potencialmente aplicáveis a qualquer problema de otimização, exigindo, para isso, apenas algumas definições no nível mais baixo, notadamente o fornecimento de um conjunto de heurísticas simples que lidam diretamente com o problema a ser resolvido. O objetivo primordial das hiper-heurísticas é alcançar resultados consideravelmente superiores aos que seriam obtidos pela aplicação de heurísticas triviais, como, por exemplo, métodos baseados em apenas uma das heurísticas de baixo nível. Em outras palavras, uma hiper-heurística deve ser capaz de minimizar os efeitos dos erros de decisão praticados por cada uma das heurísticas de baixo nível, através da escolha das situações mais adequadas para a sua aplicação, dentro de um processo de otimização com intercalação de heurísticas.

Obviamente, nada impede que as hiper-heurísticas sejam implementadas ou utilizadas de maneira a enfatizar um único problema, aumentando a qualidade das soluções através do emprego de informações específicas na própria hiper-heurística ou da incorporação de métodos sofisticados (por exemplo, implementações bem sucedidas de meta-heurísticas) pelas heurísticas de baixo nível. Porém, com isso, despreza-se a facilidade de aplicação, um dos principais diferenciais das hiper-heurísticas.

3.1 Precursores

Pode-se dizer que a pré-história das hiper-heurísticas foi construída pelos métodos gerenciadores de heurísticas. Ross et al. descrevem essa pré-história no texto em que se baseia esta seção [53].

A utilização dinâmica de heurísticas já estava presente, na década de 1980, em métodos de planejamento automático. Nesse sentido, é possível que o exemplo mais notável seja o sistema PRODIGY ([46] apud Ross et al. [53]), que, através de aprendizagem automática, evoluía seus processos de decisão. Porém, os primeiros casos conhecidos de seleção explícita de heurísticas em múltiplos pontos de decisão somente surgiram na década de 1990. Desses, pode-se destacar

o escalonador LR-26 (parte do sistema COMPOSER [30]), utilizado no planejamento de comunicações entre satélites artificiais e três estações em terra (um problema de programação inteira binária, com centenas de variáveis e milhares de restrições).

O algoritmo utilizado pelo LR-26 dividia-se em 5 fases: definir multiplicadores de Lagrange [31], para a criação de um problema relaxado; encontrar uma solução que não necessariamente satisfaça todas as restrições; ordenar o conjunto de restrições não satisfeitas; propor algumas valorações, visando à viabilidade da solução; experimentar as valorações propostas. Para a efetiva implementação desse processo, havia 28 heurísticas disponíveis para as diferentes fases, representando 2.592 estratégias diferentes, a serem avaliadas experimentalmente. Devido à inviabilidade do processo de avaliação de todas as estratégias com os 50 casos de teste disponíveis, foi realizada uma seleção preliminar heurística, que reduziu para 51 o número de estratégias disponíveis, viabilizando o teste exaustivo. O resultado de todo esse processo de seleção foi um ganho decisivo de eficiência e eficácia, apesar da dependência do algoritmo em relação à representatividade dos casos de teste.

Em 1994, Fang et al. [20] propuseram um método (então chamado de *Evolving Heuristic Choice*, o que pode ser traduzido como "evolução das seleções de heurísticas") para a utilização de um algoritmo genético na escolha das heurísticas a serem aplicadas na resolução do problema de escalonamento conhecido como *open-shop scheduling*. Cada instância desse problema consiste em alguns *serviços*, compostos por tarefas que podem ser escalonadas em qualquer ordem. Cada tarefa deve executar em uma máquina específica, durante um tempo fixo. Sejam *m* o número de serviços e *n* o número de tarefas. No algoritmo genético desenvolvido, cada cromossomo possuía *n-1* pares de genes, de forma que cada *i*-ésimo par (s_i, h_i) indicava o que seria realizado na *i*-ésima iteração: selecionar o s_i-ésimo serviço incompleto (em uma lista circular) e escolher, utilizando a heurística h_i, uma de suas tarefas para ser colocada na primeira posição possível da agenda. Algumas das heurísticas utilizadas eram: selecionar a tarefa com menor tempo de processamento; selecionar a tarefa com maior tempo de processamento; dentre as tarefas que podem ser escalonadas na primeira posição livre, selecionar a que tenha maior tempo de processamento.

A estratégia de Fang et al. obteve excelentes resultados em diversas instâncias conhecidas, incluindo, em alguns casos, soluções superiores a todas as existentes. Porém, há duas características indesejáveis, estando uma delas relacionada ao consumo de recursos (o tamanho dos cromossomos cresce linearmente com o número de tarefas) e outra à generalidade (o significado dos cromossomos está atrelado ao problema *open-shop scheduling*). Observe que o algoritmo a seguir ainda sofre desses males.

Em 1998, Hart et al. [36] lidaram com outro problema de escalonamento – desta vez, um

caso real. O problema consistia no transporte de galinhas vivas das fazendas às fábricas de processamento, de forma a atender aos pedidos de revendedores, que poderiam ser alterados todos os dias. Mais especificamente, a tarefa era organizar, sempre que necessário, o trabalho dos veículos disponíveis para transporte. Algumas das muitas restrições envolvidas eram: as galinhas não podem ser entregues muito antes dos horários sugeridos pelas fábricas; os veículos estão disponíveis em diferentes dias e horários e têm capacidades distintas; as fazendas não podem ser visitadas em qualquer ordem.

O tratamento dado por Hart et al. para o problema acima emprega dois algoritmos genéticos. No primeiro, cada cromossomo é composto por uma permutação dos pedidos e duas seqüências de decisões, sendo que a primeira seqüência determina a divisão dos pedidos em tarefas de transporte, enquanto a segunda aloca veículos para cada uma dessas tarefas. O papel do segundo algoritmo genético é, partindo das associações criadas no passo anterior, estipular os horários de chegada dos veículos às fábricas. Na terceira e última fase, o algoritmo determina horários de partida para os veículos. Os resultados obtidos pelo algoritmo foram satisfatórios para a resolução do problema real.

Um ano depois, Terashima-Marín et al. [59] desenvolveram um algoritmo genético cujos cromossomos já não tinham seus tamanhos incrementados pelo crescimento das instâncias. O problema tratado foi a construção de grades horárias para exames acadêmicos, levando em conta diversas restrições. Os exames podem ocorrer em alguns intervalos de tempo fixos e existem salas de vários tamanhos. Dois exames não podem ocorrer ao mesmo tempo se algum estudante realizará ambos. Dois exames podem ocorrer na mesma sala, ao mesmo tempo, se a sala tiver capacidade suficiente. Alguns exames têm suas próprias restrições de horários. É desejável que um aluno nunca realize dois exames consecutivos e que os exames de maior porte ocorram antes.

O algoritmo elaborado por Terashima-Marín et al. tinha como objetivo definir estratégias para um outro algoritmo, que lidava diretamente com o problema em duas fases. A primeira etapa desse algoritmo específico repete, enquanto uma certa condição X não for satisfeita, a aplicação de uma heurística H_1 para selecionar um evento, seguida por uma heurística H_2, que determina um horário para a realização do evento escolhido. A segunda etapa se assemelha à anterior, exceto pela condição de parada e pela substituição das heurísticas H_1 e H_2 por, respectivamente, heurísticas H_3 e H_4. O algoritmo genético é responsável por selecionar as quatro heurísticas acima, além da condição de parada X. As *soft constraints* (expressão que pode ser traduzida como “restrições desejáveis” – ou simplesmente “preferências” – e que representa elementos muitos comuns em problemas de otimização) são tratadas pelas heurísticas de baixo nível. A aptidão de cada cromossomo é medida através da construção direta da grade horária a que ele

dá origem. Apesar desta última característica, o algoritmo foi capaz de encontrar soluções extremamente satisfatórias, mesmo em instâncias de grande porte. De fato, uma posterior geração exaustiva de cromossomos não produziu resultados muito superiores aos anteriormente obtidos.

3.2 Resumo dos Métodos Existentes

Umas das principais hiper-heurísticas existentes é o algoritmo de Soubeiga, Cowling e Kendall (Seção 3.3), que se baseia em uma *função de decisão* para, em cada iteração, selecionar e utilizar uma heurística. O método foi desenvolvido em 2000 e, desde então, sofreu diversas modificações. Um tipo distinto de hiper-heurística, também bastante trabalhado desde sua primeira implementação, é o algoritmo genético desenvolvido em 2002 por Cowling, Kendall e Han. Descrevemos essa hiper-heurística na Seção 3.4.

Burke et al. [9] apresentaram uma hiper-heurística bastante simples, baseada em uma função de decisão que emprega uma estratégia de aprendizado por reforço (*reinforcement learning* [58]), além de uma lista tabu (de heurísticas), na seleção de uma heurística em cada iteração. O método foi aplicado a um problema real de escalonamento de enfermeiras em um hospital britânico de grande porte e a um problema de criação de grades horárias de aulas universitárias. A hiper-heurística obteve resultados comparáveis aos produzidos por heurísticas mais específicas, além de se mostrar mais robusta que estas e capaz de construir soluções viáveis com grande desenvoltura.

Em 2003, Burke et al. [8] apresentaram uma hiper-heurística baseada em um sistema de formigas artificiais. No grafo percorrido pelas formigas, cada vértice representa uma heurística, de forma que a passagem por uma aresta (u,v) indica a aplicação da heurística representada pelo vértice *v* logo após a utilização da heurística representada por *u*. O método foi aplicado, em diversas variantes, a um problema de escalonamento de apresentações de projetos, com o auxílio de uma heurística construtiva para o fornecimento da solução inicial. A hiper-heurística obteve resultados superiores a hiper-heurísticas mais simples, demonstrando que seus mecanismos foram capazes de selecionar seqüências de heurísticas com maior impacto positivo.

Bai e Kendall [2] desenvolveram o método de recozimento simulado como uma hiper-heurística, de forma que as heurísticas de baixo nível cumprissem o papel dos operadores naquela meta-heurística. A hiper-heurística foi aplicada a 7 instâncias de um problema de geração automática de planogramas de loja, com o auxílio de uma heurística construtiva bastante simples para a criação da solução inicial. O método obteve resultados superiores aos produzidos pela hiper-heurística de Soubeiga, Cowling e Kendall (Seção 3.3), além de se mostrar melhor que outras hiper-heurísticas mais simples.

Em 2002, Ross et al. elaboraram um método hiper-heurístico [54] para combinar algoritmos de aproximação e heurísticas construtivas na resolução de um problema de empacotamento (*bin-packing*) unidimensional. O algoritmo emprega um classificador (treinado durante as execuções da hiper-heurística) cujo papel é, em cada iteração, selecionar uma heurística (idealmente, a melhor) para aplicar à solução parcial corrente. Tanto a hiper-heurística quanto cada uma das heurísticas de baixo nível foram aplicadas a centenas de instâncias conhecidas e a hiper-heurística foi capaz de superar consideravelmente todos os outros métodos, no que diz respeito ao número de instâncias para as quais foram encontradas soluções com os mesmos custos das melhores soluções conhecidas. Note, porém, que o método de Ross et al. emprega mais informações específicas que as outras hiper-heurísticas citadas nesta seção, uma vez que o classificador precisa analisar as soluções.

3.3 Uma Hiper-Heurística de Busca por Entornos

Em sua tese de doutorado [57], publicada em 2003, Soubeiga apresenta duas hiper-heurísticas baseadas em funções de decisão elaboradas. Um desses métodos consiste no recozimento simulado, implementado como hiper-heurística, enquanto o outro utiliza uma função de decisão desenvolvida pelo autor. Ademais, Soubeiga construiu diversas hiper-heurísticas triviais, o que permitiu confirmar o impacto positivo do uso de estratégias cuidadosas na seleção de heurísticas. O autor empregou seus métodos na resolução de três problemas de escalonamento de pessoal. A seguir, descrevemos duas dessas aplicações [15, 17], de forma a apresentar algumas variantes do método mais original de Soubeiga, cuja evolução pode ser acompanhada em diversos artigos publicados por Soubeiga et al. [14, 15, 17, 16].

3.3.1 Problema da Feira Comercial

O primeiro problema abordado na tese de Soubeiga consiste na definição de uma agenda para um evento comercial que envolve *fornecedores* – representantes de empresas interessadas em vender produtos ou prestar serviços – e *delegados* – representantes de empresas potencialmente interessadas em produtos e serviços. Cada fornecedor paga uma taxa de participação e indica uma lista de delegados que gostaria de encontrar, classificando cada possível encontro como *prioritário* ou *não-prioritário*. As empresas interessadas em enviar delegados não pagam taxa de inscrição, mas nem todos esses potenciais delegados são incluídos no evento. Além disso, com o objetivo de promover a interação entre os delegados, organizam-se seminários. Cada delegado indica os seminários de que gostaria de participar e, caso seja convidado para o evento, tem sua participação garantida em todos eles. Cada encontro ocupa uma vaga na agenda e cada

seminário ocupa três. Na instância real tratada, havia 43 fornecedores, 99 potenciais delegados, 12 seminários e 24 vagas disponíveis na agenda. O objetivo era escalonar os encontros, ou seja, criar tuplas do tipo *(delegado, fornecedor, horário)*, satisfazendo as seguintes restrições (as duas últimas são apenas desejáveis):

- Cada delegado deve participar de no máximo 12 encontros.
- Nenhum delegado pode participar de duas atividades (seminário ou encontro) ao mesmo tempo.
- Nenhum fornecedor pode participar de dois encontros ao mesmo tempo.
- Cada fornecedor deve ter ao menos 17 encontros prioritários.
- Cada fornecedor deve ter ao menos 20 encontros.

O valor da função objetivo, a ser minimizado, no problema acima é calculado com a expressão $B + 0,05C + 8D$, onde B é a soma das penalidades associadas ao não cumprimento da penúltima restrição, C é a soma das penalidades relacionadas à última restrição e $D = d - 72$, sendo d o número de delegados convidados para o evento e 72 um limite inferior para o número de delegados necessários nas soluções em que não há violações da última restrição. A importância de D se deve ao fato de que a presença dos delegados implica gastos, para a companhia organizadora da feira, com hospedagem e transporte. Cowling et al. apresentaram duas versões de uma hiper-heurística [14, 15] para o tratamento dessa instância. Abaixo, descrevemos a versão menos dependente de parâmetros [15].

O algoritmo guloso desenvolvido pela companhia organizadora obteve uma solução de custo $444,43$, enquanto um outro algoritmo guloso, elaborado pelos responsáveis pela hiper-heurística, resultou em uma solução de custo $225,55$, que foi utilizada como ponto de partida para a hiper-heurística. Esta última foi implementada de forma a receber um conjunto de funções componentes da função objetivo (neste caso, B, C e D), incluindo os respectivos pesos (1, $\frac{5}{100}$ e 8, para a instância abordada). A hiper-heurística executa um processo iterativo que, em cada passo, seleciona aleatoriamente uma função componente c (com probabilidades proporcionais aos pesos das funções) e decide, desconsiderando os demais critérios, qual heurística de baixo nível deve ser aplicada. Seja H_k a heurística executada na iteração anterior à corrente. A seleção da heurística seguinte se baseia em três fatores históricos: o desempenho de cada heurística H em relação a c ($f_{1c}(H)$); o desempenho de cada heurística H, em relação a c, quando executada logo após H_k ($f_{2c}(H, H_k)$); o tempo de CPU decorrido desde que cada heurística H foi executada pela última vez ($f_3(H)$). Cada fator histórico possui um peso dinâmico. Mais especificamente, a função de decisão para cada função componente c pode ser definida como:

$f_c(H_i) = \alpha f_{1c}(H_i) + \beta f_{2c}(H_i, H_k) + \gamma f_3(H_i)$, para toda heurística H_i.

Claramente, as funções f_{1c} e f_{2c} podem ser vistas como fatores de intensificação, enquanto f_3 pode ser vista como um fator de diversificação. Em cada iteração, observa-se o resultado da utilização da heurística selecionada. Caso o custo da solução diminua, o fator de intensificação que, multiplicado por seu peso, constituir o maior termo da função de decisão terá seu peso elevado em um número que é proporcional ao ganho de qualidade. Na situação oposta, o peso do mesmo fator é reduzido. O fator de diversificação tem seu peso elevado quando, durante um certo número de iterações, não há diminuição de custo. De forma análoga, o peso desse fator é decrementado quando há seguido aumento de qualidade.

O caráter dinâmico dos pesos α, β e γ é também o responsável por evitar que as diferenças entre os valores dos termos componentes de f_c sejam demasiadamente influenciadas pelo tipo de medida presente em cada fator histórico. Como ilustração, considere um exemplo em que as heurísticas de baixo nível tenham um tempo médio de execução de 100 segundos (sendo f_3 um indicador medido em segundos) e a função objetivo sempre forneça valores no intervalo $[-10, 10]$. Claramente, nesse exemplo, uma especificação descuidada dos pesos provavelmente faria com que o termo que depende de f_3 fosse constantemente decisivo na escolha da heurística a ser executada.

Foram desenvolvidas 10 heurísticas de baixo nível para realizarem as operações diretas sobre as soluções. Abaixo, apresentamos brevemente o passo realizado por cada uma dessas heurísticas. Note que tais operações assemelham fortemente as heurísticas de baixo nível aos operadores utilizados por métodos de busca por entornos.

- Remover um delegado.
- Duas diferentes heurísticas para: selecionar um delegado, remover um de seus encontros e adicionar outro.
- Adicionar um delegado.
- Adicionar um encontro para um fornecedor insatisfeito.
- Adicionar um encontro para um fornecedor que esteja insatisfeito em relação aos seus encontros prioritários.
- Remover encontros excedentes de um fornecedor.
- Remover um encontro e adicionar outro para um fornecedor.

- Duas diferentes heurísticas para: selecionar um fornecedor r que esteja insatisfeito em relação a seus encontros prioritários, remover um encontro de r e adicionar outro.

Para fins de comparação, foram desenvolvidas 4 hiper-heurísticas triviais. A seguir, descrevemos como cada uma dessas hiper-heurísticas funciona em uma iteração:

- *SimpleRandom*: seleciona uma heurística aleatoriamente e a aplica à solução corrente.
- *RandomDescent*: seleciona uma heurística aleatoriamente e a aplica repetidamente até que não haja ganho de qualidade.
- *RandomPerm*: cria aleatoriamente uma permutação das heurísticas e a aplica.
- *RandomPermDescent*: semelhante ao item anterior, mas cada heurística da permutação é aplicada, repetidamente, até que não haja ganho de qualidade.

Cada uma das hiper-heurísticas desenvolvidas foi executada 10 vezes, com 30 minutos para cada execução. Como resultado de cada método, considerou-se a média de custo das 10 soluções fornecidas por ele. As duas hiper-heurísticas simples que aplicam cada heurística repetidamente foram capazes de melhorar significativamente a qualidade da solução inicial, obtendo custo em torno de 63. Esse resultado motivou a implementação da hiper-heurística principal de maneira que cada heurística selecionada como conseqüência dos fatores de intensificação fosse utilizada repetidamente, até não mais obter diminuição de custo. Com isso, o método principal superou decisivamente os demais, atingindo um custo médio de $47,65$.

3.3.2 Problema de Escalonamento de Apresentações

Como demonstração de robustez, a hiper-heurística descrita acima foi posteriormente aplicada [17] a um problema de escalonamento de apresentações modelado para uma universidade britânica. Com menos de três semanas de desenvolvimento, foram obtidas soluções de qualidade. No contexto considerado, cada aluno no último ano da graduação deve desenvolver um projeto, supervisionado por um docente. Tal projeto deve ser apresentado para uma banca composta por três docentes (possivelmente incluindo o próprio supervisor): *primeiro avaliador*, *segundo avaliador* e *observador*. As apresentações são organizadas por sessão e cada par *(sessão, sala)* deve conter, no máximo, 6 apresentações. O objetivo é construir tuplas na forma *(aluno, avaliador1, avaliador2, observador, sessão, sala)*, sob a restrição de que nenhum docente pode, em uma mesma sessão, compor banca em salas distintas. O valor a ser minimizado pode ser expresso como $0,5A+B+0,3C-D$, onde:

- *A* é a variância do número de apresentações por docente.
- *B* é a variância do número de sessões por docente.
- *C* é a variância do número de sessões “ruins” por docente. Uma sessão é “ruim” se ocorre antes das 10h ou após as 16h.
- *D* é um número que cresce com a quantidade de apresentações em que o supervisor faz parte da banca e com o grau de interesse de cada docente na área de cada projeto para que compõe banca.

Na instância considerada, existem 26 docentes e apenas duas salas disponíveis para 80 sessões, que devem comportar 151 apresentações. Os responsáveis pela hiper-heurística desenvolveram uma heurística construtiva razoavelmente simples e utilizaram a solução obtida como ponto de partida. Foram implementadas 8 heurísticas de baixo nível, fornecidas às 4 hiper-heurísticas triviais anteriormente implementadas e a 3 versões da hiper-heurística principal:

HH1: a hiper-heurística aplicada ao problema da feira comercial.

HH2: semelhante a *HH1*, mas utiliza pesos fixos para os fatores históricos ($\alpha = \beta = 0,1$ e $\gamma = 2,5$).

HH3: semelhante a *HH1*, mas não utiliza as funções componentes da função objetivo, ou seja, considera um único critério (a própria função objetivo) ao selecionar heurísticas.

Como resultado de cada hiper-heurística, considerou-se o custo médio obtido em 10 execuções de 10 minutos. Todas as hiper-heurísticas melhoraram consideravelmente a solução inicial, obtendo qualidade muito superior à de uma solução manual. O algoritmo *HH3* foi o único a se mostrar decisivamente melhor que as hiper-heurísticas triviais. Outros resultados observados foram:

- Contrariando as expectativas, *HH2* mostrou-se superior a *HH1* na instância considerada.
- A freqüência de utilização das heurísticas de baixo nível não é uniforme entre as versões da hiper-heurística principal.
- Quando partem da solução manual, as hiper-heurísticas desenvolvidas encontram soluções muito inferiores às obtidas anteriormente, o que sugere a existência de uma fragilidade nesses métodos.
- No problema da feira comercial, *HH1* obteve o melhor desempenho, o que pode indicar uma dependência relevante em relação ao número de componentes da função objetivo.

3.4 Um Algoritmo Genético Hiper-Heurístico

Em 2002, Cowling et al. [13] elaboraram um algoritmo genético em que cada cromossomo é uma seqüência de heurísticas de baixo nível. Posteriormente, o método sofreu diversas alterações, incluindo a adaptabilidade nos tamanhos dos cromossomos [34], a classificação de genes como tabu [32] e a reelaboração do processo de inserção e remoção de heurísticas [33]. Esta última versão também foi aplicada ao problema de escalonamento de apresentações que descrevemos acima, obtendo resultados ligeiramente superiores aos de Soubeiga, em um tempo de execução 50% maior. A seguir, descrevemos as primeiras versões do método de Cowling et al. [13, 34].

3.4.1 Problema de Escalonamento de Professores

Os autores lidam com 5 instâncias de um problema de escalonamento, em que são dados:

- 25 professores.
- 10 centros de treinamento, que também chamaremos de *localidades*. Cada centro possui um número limitado de salas.
- Uma matriz que, a cada par (d,l), sendo d um professor e l uma localidade, associa um valor de penalidade, indicando quão indesejável é que d precise ministrar um curso em l.
- 60 vagas em uma grade de horários.
- Um conjunto de cursos. A cada curso estão associados:
  - Um conjunto de professores competentes para ministrar o curso.
  - Um intervalo dentro do qual deve estar contido o início da execução do curso.
  - Um conjunto de localidades dentre as quais deve ser selecionado o centro de treinamento onde o curso será dado.
  - A duração do curso – um inteiro em $[1,5]$, indicando o número de vagas na grade horária – e um valor que indica a sua prioridade (quanto maior este valor, maior é a importância de se ministrar o curso).

Seja W a soma das prioridades dos cursos realizados e seja D a soma das penalidades aplicadas. O objetivo é maximizar o valor de $W - D$, satisfazendo as seguintes restrições adicionais:

- Um professor não pode ministrar dois cursos simultaneamente.

- Nenhum professor pode trabalhar em mais de 60% das vagas de horários disponíveis.

Para fins de comparação, foram implementados dois algoritmos baseados em meta-heurísticas, operando diretamente sobre o problema. O primeiro desses métodos é um algoritmo genético em que cada cromossomo possui 25 genes, associados aos professores. Cada gene contém todas as informações associadas ao respectivo professor no escalonamento representado pelo cromossomo. O operador de cruzamento seleciona aleatoriamente dois genes e permuta os conjuntos de atividades entre os dois respectivos professores, resolvendo, em seguida, as possíveis inviabilidades. O operador de mutação seleciona um gene aleatoriamente e acrescenta uma atividade à lista do respectivo professor. A população inicial é obtida a partir dos escalonamentos criados por um algoritmo de *hill-climbing*. Os valores mais adequados para a taxa de cruzamento ($0,6$), a taxa de mutação ($0,03$), o tamanho da população (30) e o número de gerações (100) foram determinados empiricamente. Na passagem entre duas gerações, os 10 melhores indivíduos são sempre mantidos – estratégia comumente chamada de *elitismo*.

O segundo método específico implementado é um algoritmo memético, que estende o algoritmo genético descrito acima. A busca local é aplicada em cada geração, após as operações de cruzamento e mutação. Os operadores utilizados – os mesmos fornecidos à hiper-heurística – podem ser classificados em três tipos: *heurísticas de adição*, *heurísticas de intercâmbio* e *heurísticas de adição-remoção*. Cada uma das 5 heurísticas de adição utiliza uma diferente estratégia para escalonar um curso. As 4 heurísticas de intercâmbio funcionam de maneira semelhante às heurísticas de adição, mas, na presença de inviabilidade, podem trocar cursos de posição. As 3 heurísticas de adição-remoção consideram os cursos não escalonados em ordem decrescente de prioridade e escalonam o primeiro que for possível. Um curso selecionado é escalonado quando não causa conflitos com outros cursos ou quando, mesmo com a remoção dos cursos conflitantes, resulta em um aumento na qualidade da solução.

O processo de otimização local utilizado pelo algoritmo memético seleciona um curso em cada iteração e procura escaloná-lo com uma heurística de adição aleatoriamente escolhida. Caso a heurística não obtenha sucesso, seleciona-se aleatoriamente uma heurística de intercâmbio. Se ainda não for possível escalonar o curso, utiliza-se uma heurística de adição-remoção (também escolhida de forma aleatória). O processo estará concluído quando todos os cursos houverem sido considerados para escalonamento.

No algoritmo genético hiper-heurístico, cada cromossomo é uma seqüência de inteiros do intervalo $[0,11]$, de forma que cada gene representa uma das 12 heurísticas de baixo nível citadas acima. A avaliação de um cromossomo ocorre através da aplicação de sua seqüência de heurísticas à melhor solução já encontrada. Utilizou-se cruzamento de ponto único e, após alguns

experimentos, foram adotadas uma taxa de cruzamento de $0,6$ e uma taxa de mutação de $0,1$, com uma população de tamanho 30, durante 100 gerações. Para a construção da população inicial, foi adotado um processo simples que gera aleatoriamente números inteiros no intervalo $[0,11]$. É importante notar que a hiper-heurística se mostrou robusta em relação às alterações de parâmetros realizadas nos testes.

Quatro versões da hiper-heurística foram adotadas na comparação com outros métodos. As diferenças entre essas versões residem na adaptabilidade e na função de aptidão. As duas versões adaptativas elevam o valor da taxa de mutação e diminuem o valor da taxa de cruzamento quando, durante três gerações seguidas, não há aumento na aptidão média da população. Essas versões reduzem a taxa de mutação e aumentam a taxa de cruzamento quando há seguido aumento de qualidade em três gerações. A elevação de valor, em todos os casos, segue a operação $A = \frac{A+1}{2}$, sendo A o parâmetro considerado. A diminuição ocorre através de uma redução à metade. Duas versões da hiper-heurística adotam como aptidão para cada indivíduo o valor objetivo da solução encontrada pela aplicação da seqüência de heurísticas sugerida pelo cromossomo (note que, como no problema considerado o objetivo é maximizar uma função, pode-se utilizar o valor objetivo diretamente como aptidão). As outras duas versões levam a eficiência em consideração, dividindo o valor objetivo pelo tempo de CPU consumido pela aplicação das heurísticas.

Os testes realizados comparam os resultados de diversos métodos, dentre os quais destacamos:

- As 4 hiper-heurísticas.
- As 12 heurísticas de baixo nível. Cada uma delas foi executada repetidamente até chegar ao ponto em que já considerou todos os cursos ainda não escalonados.
- O algoritmo genético específico.
- O algoritmo memético.
- Cinco heurísticas ($H1$ a $H5$) que foram os respectivos resultados de cinco execuções da hiper-heurística mais simples – a versão não adaptativa cuja função de aptidão não leva em conta o tempo de CPU – em uma instância razoavelmente complexa do problema. Cada uma dessas cinco heurísticas corresponde, portanto, a um cromossomo do tipo utilizado pelas hiper-heurísticas.

As 5 instâncias utilizadas diferem principalmente no número médio de professores que podem ministrar cada curso e no número médio de localidades em que cada curso pode ser dado. Na instância mais complexa, há, para cada curso, apenas uma localidade e no máximo cinco

professores disponíveis. Na instância mais simples, cada curso pode ser ministrado por qualquer professor, em qualquer localidade. As instâncias são baseadas em um caso real ocorrido em uma instituição financeira de grande porte. As observações mais importantes, no que diz respeito aos resultados, foram:

- As heurísticas $H1$ a $H5$ se mostraram superiores às 12 heurísticas mais simples, em todas as instâncias do problema, além de se mostrarem comparáveis ao algoritmo genético direto e ao algoritmo memético. Este resultado indica que a hiper-heurística explorou as qualidades das heurísticas simples de maneira satisfatória. Mais importante: esta observação reforça a hipótese de que a qualidade de uma seqüência de heurísticas não depende excessivamente da solução à qual ela é aplicada.

- Naturalmente, as soluções encontradas pelas 12 heurísticas simples são muito inferiores aos resultados das quatro versões da hiper-heurística – que, no entanto, demandam um tempo de CPU muito maior.

- Ambas as meta-heurísticas que operam diretamente sobre o problema forneceram, em maior tempo de CPU, resultados consideravelmente inferiores aos das hiper-heurísticas.

- A freqüência de utilização das heurísticas de baixo nível não é uniforme entre as versões da hiper-heurística (note que resultado similar houvera sido observado na experimentação do método de Soubeiga).

- A hiper-heurística mais simples obteve os melhores resultados em todas as instâncias, excetuando-se a menos complexa – esta foi tratada com maior desenvoltura pela hiper-heurística que utiliza parâmetros adaptativos e adota a própria função objetivo como função de aptidão.

Em outro artigo [34], os autores relatam uma mudança que torna suas hiper-heurísticas adaptativas em relação aos tamanhos dos cromossomos. O objetivo desta nova estratégia é óbvio: aumentar a flexibilidade e a autonomia das hiper-heurísticas, permitindo-lhes manipular seqüências de heurísticas utilizando procedimentos menos restritos. Idealmente, os tamanhos mais adequados para os cromossomos serão determinados sem a necessidade de se realizarem diversos experimentos. Na nova estratégia, explora-se ainda mais a hipótese de que a qualidade de uma seqüência de heurísticas não depende decisivamente da solução à qual ela é aplicada. Mais especificamente, foram introduzidos, sem a eliminação dos antigos operadores de cruzamento e mutação, três novas operações para a fase de reprodução:

- Um novo operador de cruzamento seleciona a melhor seqüência de genes – aquela que causa maior diminuição de custo – de cada cromossomo e permuta essas seqüências entre os dois cromossomos. Essa operação foi batizada pelos autores como cruzamento *best-best*.

- Uma nova mutação seleciona a pior seqüência de genes – a que tem o pior impacto na função objetivo ou a mais longa dentre as que não promovem aumento de qualidade – e a remove do cromossomo.

- Outra mutação seleciona a melhor seqüência de genes de um cromossomo aleatoriamente escolhido e a insere, em uma posição aleatória, no cromossomo selecionado para sofrer mutação.

Devido à presença dos novos operadores, as hiper-heurísticas utilizam um valor de penalidade para evitar a presença de cromossomos demasiado grandes. Cromossomos mais penalizados terão menor probabilidade de serem selecionados para recombinação. A penalidade é calculada pela expressão $\frac{TC}{M}$, onde T é o tamanho do cromossomo, C é o tempo de CPU necessário para a avaliação do cromossomo e M é o aumento de qualidade resultante da aplicação da seqüência de heurísticas sugerida pelo cromossomo. Foram criadas quatro novas versões da hiper-heurística, através do acréscimo dos novos operadores às quatro versões outrora implementadas. Além disso, foi experimentada uma versão, com tamanho adaptativo de cromossomo, que não utiliza o novo operador de cruzamento. Dentre os resultados observados, destacamos:

- Cada versão da hiper-heurística com tamanho adaptativo de cromossomo superou, na maioria dos casos, a respectiva versão com cromossomos de tamanho fixo.

- A versão vencedora da hiper-heurística é justamente a nova versão da hiper-heurística que fornecera as melhores soluções no estudo anterior.

- A versão que não utiliza o novo operador de cruzamento obteve resultados comparáveis (embora inferiores) aos das demais hiper-heurísticas, com consumo de tempo ligeiramente menor.

4 Aspectos Investigados

A diferença mais básica entre os dois métodos hiper-heurísticos detalhados no Capítulo 3 reside no fato de que o método de Soubeiga se baseia em um processo de busca por entornos, enquanto a estrutura geral do método de Cowling et al. advém de uma meta-heurística populacional. Note, porém, que há uma importante distinção entre os algoritmos genéticos de Cowling et al. e as implementações mais comuns dessa meta-heurística, uma vez que a abordagem hiper-heurística lida com uma população de algoritmos. A operação do método de Soubeiga, por sua vez, não se distingue tão fortemente das estratégias de busca por entornos mais tradicionais. De fato, nesta última comparação, chama a atenção apenas o tratamento das soluções, dos operadores e da função de avaliação como "caixas-pretas", característica comum nas hiper-heurísticas.

A maioria dos métodos hiper-heurísticos existentes se assemelham ao método de Soubeiga (por serem processos de busca por entornos que aplicam operadores apenas às soluções do problema original [57, 2, 9, 14, 16, 17, 15]) ou aos algoritmos genéticos de Cowling et al. (caso dos métodos evolutivos cujas populações são compostas por algoritmos [8, 13, 34, 32, 33, 59]). De fato, ao considerarmos a implementação de hiper-heurísticas como processos de busca por entornos ou algoritmos populacionais, quatro possibilidades imediatamente chamam a atenção:

1. A evolução de uma população composta por soluções do problema original.
2. A evolução de uma população composta por algoritmos que façam parte de algum processo de resolução do problema original.
3. A execução de um processo de busca por entornos que percorra diretamente o espaço das soluções do problema original.
4. A execução de um processo de busca por entornos que percorra um espaço de algoritmos.

Como vimos, as estratégias sugeridas nos itens 2 e 3 já conheceram seus primeiros estágios e se apresentam como caminhos promissores. Por sua vez, a abordagem indicada no item 1 não é,

aparentemente, explorável no contexto das hiper-heurísticas, uma vez que os métodos populacionais mais conhecidos realizam operações que exigem algum conhecimento da estrutura dos indivíduos. O item 4, por outro lado, configura uma opção perfeitamente plausível, sendo uma via natural para exploração quando as duas principais estratégias expostas acima alcançarem um estágio de maior maturidade.

Naturalmente, a implementação de hiper-heurísticas não está restrita às possibilidades apresentadas acima. Porém, desenvolver métodos inspirados em meta-heurísticas de qualidade prática reconhecida é provavelmente a maneira mais segura de conduzir experimentos no campo das hiper-heurísticas. Ademais, essa forma de exploração é ainda uma contribuição ao estudo das meta-heurísticas. De fato, em alguns casos, uma hiper-heurística pode ser extremamente semelhante a uma implementação tradicional de determinada meta-heurística. Note, por exemplo, como o recozimento simulado pode ser trivialmente implementado como hiper-heurística, bastando, para isso, que os operadores, a função de avaliação e o procedimento de construção da solução inicial não sejam implementados dentro do corpo algorítmico correspondente ao método de recozimento simulado.

Uma hiper-heurística, ao contrário das meta-heurísticas em geral, é um algoritmo bem definido, que recebe como parâmetros as estruturas e as abstrações diretamente relacionadas ao problema a ser resolvido. Essa distinção, no entanto, só se torna relevante quando a construção das informações específicas requer um esforço reduzido. Esse fato pode tornar complexa a construção de hiper-heurísticas de busca direta por entornos (item 3, acima) que se baseiem em meta-heurísticas como a busca tabu e a busca local guiada, uma vez que a qualidade desses métodos depende de um uso intenso de informações específicas (nesse sentido, o recozimento simulado pode ser visto como uma exceção). Devido a esses empecilhos, pareceu-nos mais natural expandir o panorama das hiper-heurísticas através, principalmente, da criação de hiper-heurísticas evolutivas indiretas, de forma a contribuir de maneira importante para a ampliação das bases para o estudo de hiper-heurísticas, aproximando essas bases de uma maior estabilidade.

Os algoritmos originais mais importantes desenvolvidos neste trabalho são duas hiper-heurísticas baseadas em meta-heurísticas populacionais. Como veremos adiante, um desses métodos claramente busca inspiração na reconexão de caminhos, enquanto o outro pode ser visto como um algoritmo de estimação de distribuição hiper-heurístico. Adicionalmente, apresentamos uma nova hiper-heurística de busca por entornos, bastante simples, que foi desenvolvida para que pudéssemos avaliar até que ponto a sofisticação é importante na criação de uma hiper-heurística, uma vez que tais métodos lidam com uma quantidade bastante reduzida de informações sobre o problema em foco. Cabe notar que, diverso do que foi analisado por Soubeiga et al. (o desempenho de hiper-heurísticas com funções de decisão completamente aleatórias

que não utilizam informações sobre o histórico da busca – Seção 3.3), o aspecto que aqui procuramos avaliar é a capacidade de uma hiper-heurística cuja função de decisão baseia-se no histórico da busca, embora execute suas escolhas da maneira mais simples que pudemos conceber.

O segundo conjunto de implementações do presente trabalho tem como foco as hiper-heurísticas apresentadas nas seções 3.3 e 3.4. Foram implementadas a hiper-heurística de busca por entornos de Soubeiga (Seção 3.3) e a hiper-heurística evolutiva indireta de Cowling et al. (Seção 3.4), para que em seguida fossem experimentadas determinadas alterações, cujo impacto avaliamos com o objetivo de determinar se certas estratégias tendem a prejudicar as hiper-heurísticas ou a aumentar sua qualidade. A implementação do método de Soubeiga baseou-se principalmente na Seção 5.3.2 de sua tese [57] e, por isso, difere ligeiramente dos algoritmos descritos em seus artigos (e apresentados na Seção 3.3), enquanto o algoritmo genético hiper-heurístico de Cowling et al. foi implementado de acordo com informações colhidas dos artigos relacionados a ele [13, 34, 32, 33] e baseia-se principalmente na segunda versão apresentada pelos autores [34].

Ao desenvolvermos duas hiper-heurísticas evolutivas indiretas originais, tivemos a oportunidade de observar também o desempenho desses algoritmos em comparação com a hiper-heurística de Soubeiga, um método importante especialmente por ser uma sofisticada hiper-heurística de busca direta por entornos. Dessa maneira, demos um importante passo rumo à determinação da qualidade relativa dessas duas abordagens predominantes, que diferem significativamente entre si. Pudemos, ainda, exibir evidências de que a construção de hiper-heurísticas baseadas em meta-heurísticas criteriosamente escolhidas – de forma a não interferir nas características mais desejáveis das hiper-heurísticas – tende a oferecer resultados satisfatórios.

Todas as implementações de hiper-heurísticas realizadas neste trabalho foram cuidadosamente experimentadas, através da abordagem de dois problemas (Capítulo 5) para os quais existem diversas instâncias disponíveis publicamente e já utilizadas por muitos outros algoritmos. Com isso, pudemos não apenas adotar uma significativa quantidade de instâncias, como também garantir a relevância da experimentação com tais instâncias. Portanto, é razoável afirmar que os resultados aqui apresentados são consideravelmente mais consistentes que aqueles encontrados na grande maioria dos trabalhos já publicados que abordam hiper-heurísticas como foco principal.

Ainda no que diz respeito aos experimentos realizados, cabe notar que os dois problemas aqui adotados diferem significativamente entre si, especialmente se levarmos em conta a grande predominância dos problemas de escalonamento no panorama das hiper-heurísticas exibido nos capítulos anteriores. A robustez das hiper-heurísticas é, portanto, uma das qualidades que tive-

mos maior preocupação em observar, uma vez que, a despeito da grande importância atribuída de forma unânime a essa característica, pode-se notar que os trabalhos existentes não procuram, em geral, avaliá-la com proporcional atenção. A robustez é uma qualidade crucial, por ser capaz de tornar uma hiper-heurística apta a cumprir o seu principal papel: a reutilização simples, com resultados de boa qualidade. Naturalmente, a reutilização satisfatória muitas vezes não será trivial, considerando-se que freqüentemente será difícil desenvolver um bom conjunto de heurísticas de baixo nível. Porém, é improvável que essa dificuldade seja maior que a encontrada quando se procura atacar o problema considerado utilizando alguma outra técnica.

Para evidenciar a completa independência, em relação a problemas de otimização específicos, de uma hiper-heurística adequadamente implementada, tomamos o cuidado de desenvolver cada hiper-heurística como uma biblioteca, na linguagem C. Tal biblioteca disponibiliza uma função, capaz de atacar um problema qualquer de otimização, e especifica a interface a ser respeitada para o adequado funcionamento da função de resolução. Sendo assim, não há dúvidas de que uma hiper-heurística pode ser plenamente implementada sem estar atrelada a um problema de otimização. Cabe notar que em hipótese alguma foram feitas alterações nas hiper-heurísticas visando à resolução de determinado problema. Cada hiper-heurística foi aplicada sem modificações a todas as instâncias adotadas de ambos os problemas considerados, para que uma legítima avaliação de robustez fosse realizada.

Durante a elaboração da interface a ser disponibilizada pelas hiper-heurísticas, notamos que, apesar das definições apresentadas no panorama das hiper-heurísticas apresentado acima, existe ao menos uma interface aparentemente mais vantajosa que as previamente propostas. Em vez de estabelecer que à hiper-heurística devem ser passadas as heurísticas de baixo nível, uma função de avaliação e uma solução inicial, especificamos um formato especial para as soluções:

```
typedef struct sol solution_t;
struct sol {
  double cost;
  void* sol;
};
```

O tipo *solution_t* é uma estrutura, em linguagem C, composta por uma solução propriamente dita do problema considerado (com o tipo *void**, de maneira que as soluções possam ser representadas em absolutamente qualquer forma) e por seu custo. As hiper-heurísticas, dessa forma, exigem apenas uma solução inicial (do tipo *solution_t*) e um conjunto de heurísticas de baixo nível capazes de receber uma solução e fornecer outra (ambas também do tipo *solution_t*). A vantagem mais evidente dessa importante mudança de interface que aqui propomos é o fato de

que ela permite ao usuário da hiper-heurística uma implementação mais livre da avaliação de soluções. Como exemplo óbvio, note que o custo da solução fornecida por uma heurística de baixo nível pode, em alguns casos, ser calculado por essa heurística de forma mais eficiente caso a avaliação considere o custo da solução recebida e as mudanças nesta efetuadas para a obtenção da solução resultante. Essa técnica foi efetivamente utilizada neste trabalho, conforme veremos na Seção 5.2.

A seguir, nas seções que concluem este capítulo, apresentamos em detalhes cada uma das hiper-heurísticas implementadas, indicando as motivações para determinadas estratégias. Nas seções 4.1 e 4.2, referentes, respectivamente, ao método de Soubeiga e ao algoritmo genético de Cowling et al., descrevemos ainda as técnicas experimentadas nessas hiper-heurísticas.

4.1 Extensões para a Hiper-Heurística de Soubeiga

Em sua tese [57], Soubeiga recomenda algumas direções para trabalhos futuros. Uma de suas idéias, bastante natural, é a armazenagem de informações sobre o desempenho de seqüências de heurísticas com tamanho maior que dois. Em outras palavras, a função de decisão adicionalmente passaria a considerar, em cada iteração, a qualidade, no histórico da busca, de cada heurística H_i, quando executada após as duas (ou mais) heurísticas que foram executadas por último. Note que essa alteração não é necessariamente uma melhoria, pois não há garantias de que o uso dessas informações adicionais – que pode ser feito de diversas maneiras e em diferentes intensidades – influenciará positivamente as decisões. Portanto, Soubeiga de fato propôs um interessante caminho para experimentação, que neste trabalho foi trilhado como veremos na Seção 4.1.2.

As avaliações numéricas da qualidade das heurísticas no histórico da busca são armazenadas em matrizes. Nas versões iniciais implementadas (conforme se pode acompanhar na Seção 4.1.1), empregamos a mesma técnica utilizada por Soubeiga para o preenchimento de tais matrizes. Em seguida, observamos os resultados decorrentes de uma modificação relatada na Seção 4.1.2: desconsiderar, ao avaliar uma heurística, o tempo de CPU utilizado em sua execução. A motivação para o descarte dessa informação está nos resultados obtidos por Cowling et al. na comparação entre as versões de seu algoritmo genético hiper-heurístico – vimos, na Seção 3.4.1, que a hiper-heurística responsável pelos melhores resultados não considera, ao avaliar um cromossomo, o tempo de execução requerido pela seqüência de heurísticas determinada por ele.

Parece consistente supor que, para a avaliação de uma heurística, é importante levar em consideração o tempo consumido, de forma a evitar que eficazes heurísticas de execução demorada sejam superestimadas. Porém, as implementações de meta-heurísticas de busca por entornos,

bem como as aplicações de hiper-heurísticas, geralmente não possuem tais discrepâncias dentro do conjunto adotado de operadores. De fato, as heurísticas empregadas consomem, na maioria dos casos, tempo bastante reduzido do ponto de vista prático. Sendo assim, parece interessante investigar se a medida do consumo de tempo não influencia de maneira inadequada a avaliação de uma heurística no caso geral.

4.1.1 Implementação da Hiper-Heurística

Como primeiro passo para possibilitar o início dos experimentos, implementamos a hiper-heurística de Soubeiga, de acordo com suas publicações [57, 14, 16, 17, 15]). Baseamo-nos especialmente na Seção 5.3.2 de sua tese de doutorado [57], de forma que o algoritmo pode ser descrito pela Figura 4.1, com o auxílio das seguintes definições:

- H_{ant} representa, durante todo o processo, a última heurística empregada pela hiper-heurística.
- f é a função de decisão (consulte a Seção 3.3), que não leva em conta possíveis decomposições da função objetivo e foi definida como a seguir:

  $f(H_i) = \alpha f_1(H_i) + \beta f_2(H_i, H_{ant}) + \delta f_t(H_i)$, para toda heurística H_i, sendo:

  * f_1 a função que mede o desempenho histórico das heurísticas.
  * f_2 a função tal que, para todo par de heurísticas (H_j, H_k), $f_2(H_j, H_k)$ representa o desempenho histórico de H_j quando executada logo após H_k.
  * f_t a função tal que, para toda heurística H_j, $f_t(H_j)$ é o tempo decorrido desde a última execução de H_j.
  * α, β e δ pesos dinamicamente ajustáveis.
- H é o conjunto das heurísticas de baixo nível e $n_h = |H|$.
- S_c é a solução corrente no processo de resolução.
- *AplicaHeuristica* é uma função que recebe uma heurística H_i e faz otimização local a partir de S_c, utilizando apenas H_i para determinação de vizinhança. Essa heurística será aplicada ao menos uma vez, mesmo que a solução produzida por ela não seja melhor que S_c. A solução final será armazenada em S_c e o valor devolvido será uma medida da diminuição total de custo produzida (note que o valor será negativo caso o custo final seja maior que o inicial).

- *NumExecucoes* é uma função que recebe uma heurística e devolve o número de vezes em que a heurística foi aplicada, desde o início da execução da hiper-heurística.
- C_I é o valor objetivo da solução inicial.

Figura 4.1: Visão Geral da Implementação.

1. Faça $IteracoesSemGanho = 0$.
2. Enquanto houver tempo disponível:
   (a) Encontre a heurística $H_{max} = argmax_{H_i \in H} \ f(H_i)$.
   (b) Se $IteracoesSemGanho \leq n_h$:
      i. Se $IteracoesSemGanho = n_h$:
         A. Encontre a heurística $H_{max\_t} = argmax_{H_i \in H - \{H_{max}\}} \ f_t(H_i)$.
         B. Faça $SolucaoArmazenada = S_c$.
      ii. Faça $F1 = \alpha f_1(H_{max})$, $F2 = \beta f_2(H_{max}, H_{ant})$ e $F3 = \delta f_t(H_{max})$.
      iii. Se $FT > F1$ e $FT > F2$:
         A. Encontre $H_{max\_1\_2} = argmax_{H_i \in H} \ \alpha f_1(H_i) + \beta f_2(H_i, H_{ant})$.
         B. Faça $Ganho_{1\_2} = AplicaHeuristica(H_{max\_1\_2})$.
         C. Se $Ganho_{1\_2} > 0$, faça $Ganho = Ganho_{1\_2}$ e diminua δ o suficiente para que $f(H_{max\_1\_2}) > f(H_{max})$.
         D. Caso contrário, faça $Ganho = AplicaHeuristica(H_{max})$.
      iv. Caso contrário:
         A. Faça $Ganho = AplicaHeuristica(H_{max})$ e obtenha o tempo de CPU $T_{H_{max}}$ gasto neste passo.
         B. Se $Ganho = 0$, faça $\varepsilon = \frac{-T_{H_{max}}}{NumExecucoes(H_{max}) \ {n_h}^2}$.
         C. Caso contrário, faça $\varepsilon = \frac{Ganho}{n_h \ |C_I|}$.
         D. Se $F1 > F2$ e $F1 > FT$, faça $\alpha = \alpha(1+\varepsilon)$.
         E. Se $F2 > F1$ e $F2 > FT$, faça $\beta = \beta(1+\varepsilon)$.
      v. Se $Ganho > 0$, faça $IteracoesSemGanho = 0$.
      vi. Caso contrário, faça $IteracoesSemGanho = IteracoesSemGanho + 1$.
   (c) Caso contrário:
      i. Aumente δ de forma que $f(H_{max\_t}) > f(H_{max})$.
      ii. Faça $S_c = SolucaoArmazenada$.
      iii. Execute $AplicaHeuristica(H_{max\_t})$.
      iv. Faça $IteracoesSemGanho = 0$.

A seguir esclarecemos alguns aspectos do algoritmo que, mesmo com a leitura da Seção 3.3, talvez não estejam suficientemente claros:

- Objetivo do item 2(b)i: caso tenham sido executadas muitas iterações sem diminuição de custo, faz sentido diversificar a busca executando a heurística H_{max_t} que não é executada há mais tempo. Tal operação não será efetuada na presente iteração, mas terá lugar na iteração seguinte, partindo da atual solução S_c, caso a iteração em execução não promova diminuição do custo da solução corrente.

- O passo 2c efetua a diversificação citada no item acima e garante que tal diversificação não será agendada novamente durante as próximas n_h iterações.

- Quando o valor de FT supera os outros dois termos de $f(H_{max})$ (passo 2(b)iii), pode-se dizer que a função f está indicando a necessidade de diversificação. Sendo assim, o algoritmo procura verificar se o momento é adequado para essa ação, ou seja, se a heurística $H_{max\_1\_2}$ (que maximiza f ao se considerarem apenas os critérios de intensificação) não é capaz de melhorar a solução corrente. Caso ocorra essa melhoria, a função f é ajustada através da diminuição de δ, com a idéia de que $H_{max\_1\_2}$ deveria ter sido escolhida como H_{max}.

- Quando, no passo 2(b)iv, a heurística H_{max} não altera o custo da solução, pode ocorrer uma redução no valor de α (ou β). Essa alteração tem como motivação a idéia de que a heurística escolhida de acordo com f não era adequada. Note, ainda, que a penalização aplicada a α ou β será proporcional ao tempo de CPU que foi "perdido" com a aplicação da heurística escolhida.

Merecem menção alguns detalhes do algoritmo implementado não explicitados na Figura 4.1:

- Quando uma heurística H_i é executada, consumindo tempo T_i e diminuindo em L_i o custo da solução, as funções f_1 e f_2 (naturalmente, implementadas como matrizes) são atualizadas da seguinte maneira:

$$f_1(H_i) = \psi_1\ f_1(H_i) + \tfrac{L_i}{T_i}, \quad \text{sendo } \psi_1 \text{ uma constante tal que } \psi_1 \in [0,1).$$
$$f_2(H_i, H_{ant}) = \psi_2\ f_2(H_i, H_{ant}) + \tfrac{L_i}{T_i}, \quad \text{sendo } \psi_2 \text{ uma constante tal que } \psi_2 \in [0,1).$$

- Note que há situações (passos 2(b)iiiC e 2(c)i) em que δ é ajustado para satisfazer uma inequação. No caso de uma equação, seria trivial encontrar um valor δ' para δ de forma a satisfazê-la. Precisamos, portanto, somar a δ (passo 2(c)i) ou subtrair dele (passo 2(b)iiiC) um valor $|\delta' - \delta| + v$, sendo v um número estritamente positivo. Soubeiga sugere apenas que v seja um "número positivo pequeno" ([57], Seção 5.3.2). Porém, o valor de v deve ser definido com cautela: um valor elevado tornaria a função de decisão demasiado

dependente de f_t, enquando um valor reduzido não provocaria o efeito desejado (observe, ainda, que somar valores de magnitudes muito diferentes pode resultar na anulação do menor valor, devido às limitações no número de *bytes* utilizados para armazená-los). Como a dificuldade na definição de v se deve ao fato de que δ pode sofrer extremas variações entre diferentes execuções da hiper-heurística, definimos v como $|\delta' - \delta|\,\phi$, sendo $\phi \in (0,1)$ uma constante cujo valor determinamos empiricamente. No passo 2(b)iiiC, devemos ainda evitar que δ se torne negativo. Para isso, reduzimos δ a $\frac{\delta'}{2}$ quando a técnica acima resulta em valor negativo.

Como vimos na Seção 3.3, Soubeiga obteve sucesso ao experimentar uma versão de sua hiper-heurística com valores fixos para os pesos (α, β e δ) dos fatores históricos. Esse fato pode levar a crer que seria importante considerar essa possibilidade durante nossos experimentos. Porém, é fácil notar que uma hiper-heurística com essa característica seria completamente inadequada para uso geral, pois a diferença de magnitude entre os valores calculados por f_t e aqueles calculados por f_1 ou f_2 pode variar intensamente não só com o problema considerado como também com as instâncias tratadas. Fixar valores determinados experimentalmente para os pesos dos fatores históricos equivaleria a uma tentativa de obter excelentes resultados em determinadas instâncias, para enorme prejuízo potencial de todos os problemas e instâncias não incluídos nos experimentos.

As constantes mais importantes utilizadas em nossa implementação e não definidas por Soubeiga tiveram seus valores ajustados empiricamente, conforme segue (note que as duas primeiras aparecem nos trabalhos de Soubeiga, mas não há relatos sobre os valores adotados):

- ψ_1: recebeu o valor $0,35$, escolhido empiricamente dentre diversos valores no intervalo $[0,2;\ 0,6]$.
- ψ_2: recebeu o valor $0,47$, após experimentos com valores no intervalo $[0,36;\ 0,6]$.
- ϕ: empregamos o valor $0,101$, após experimentarmos valores pertencentes ao intervalo $[0,001;\ 0,999]$.
- MAX_STEEP: limite superior, em uma execução de *AplicaHeuristica*, para o número de vezes em que a heurística fornecida será executada. Após alguns experimentos com valores no intervalo $[4,100]$, definimos $MAX\_STEEP = 100$. Tal limite não é necessariamente o mais adequado, uma vez que, nos experimentos realizados, nenhuma execução de *AplicaHeuristica* realizou mais de 70 iterações.

Para otimizar o tempo gasto com experimentação, impusemos a restrição de que em nenhum experimento deveria ocorrer a relação $\psi_1 \geq \psi_2$. A consistência dessa limitação ficará evidente mais adiante.

4.1.2 Alterações Experimentadas

Uma vez implementado e experimentado – conforme descrito na seção anterior – o algoritmo de Soubeiga, demos andamento ao primeiro experimento envolvendo alterações significativas, consistindo em avaliações de seqüências de heurísticas com tamanho maior que dois. Essas mudanças foram efetuadas incrementalmente, de forma que produzimos duas versões adicionais da hiper-heurística:

1. Como primeira modificação, passamos a considerar o desempenho histórico de seqüências com 3 heurísticas. A fórmula de f passa, então, a ter a estrutura abaixo, onde H_{ant-1} é a heurística utilizada dois passos antes da iteração corrente e f_3 é o novo critério de intensificação:

$$f(H_i) = \alpha f_1(H_i) + \beta f_2(H_i, H_{ant}) + \gamma f_3(H_i, H_{ant}, H_{ant-1}) + \delta f_t(H_i)$$

   Com a criação da nova matriz, representando f_3, foi necessário definir também uma nova constante (ψ_3), de modo que a atualização decorrente da execução de uma heurística H_i é efetuada da seguinte maneira, sendo T_i o tempo consumido por H_i e L_i a diminuição no custo da solução:

$$f_3(H_i, H_{ant}, H_{ant-1}) = \psi_3 \; f_3(H_i, H_{ant}, H_{ant-1}) + \frac{L_i}{T_i}$$

2. Na versão posterior da hiper-heurística, utilizamos uma matriz de 4 dimensões (além das matrizes de dimensões inferiores empregadas na versão anterior), armazenando informações sobre o desempenho histórico de cada seqüência de 4 heurísticas. A fórmula para a função de decisão sofreu alteração análoga à anterior, transformando a definição de f:

$$\begin{aligned} f(H_i) = \quad & \alpha f_1(H_i) + \beta f_2(H_i, H_{ant}) + \gamma f_3(H_i, H_{ant}, H_{ant-1}) + \\ & \zeta f_4(H_i, H_{ant}, H_{ant-1}, H_{ant-2}) + \delta f_t(H_i) \end{aligned}$$

   Devido à criação da função f_4, definimos uma constante adicional (ψ_4), empregada no novo passo de atualização de desempenho histórico:

$$f_4(H_i, H_{ant}, H_{ant-1}, H_{ant-2}) = \psi_4 \; f_4(H_i, H_{ant}, H_{ant-1}, H_{ant-2}) + \frac{L_i}{T_i}$$

Através de alguns experimentos, definimos valores para ψ_3 ($0,59$, escolhido do intervalo $[0,48,\ 0,7]$) e ψ_4 ($0,69$, obtido dentre valores no intervalo $[0,6,\ 0,82]$). Optamos por manter a relação $\psi_1 < \psi_2 < \psi_3 < \psi_4$ durante todo o período de experimentação (levando em conta, é claro, que ψ_3 e ψ_4 não estão presentes em todas as versões). A motivação para impormos essa restrição é o fato de que as posições das matrizes maiores são atualizadas com menor freqüência (uma vez que, em cada iteração, atualiza-se uma posição de cada matriz), o que tende a tornar seus valores menores, em módulo, que os presentes nas matrizes de menor porte.

Como veremos no Capítulo 6, há razões para acreditarmos que as duas novas versões da hiper-heurística apresentadas acima não representam uma evolução (no que diz respeito à qualidade das soluções resultantes) em relação à primeira versão implementada. Devido a essa observação, utilizamos a versão inicial da hiper-heurística como base para a derradeira experimentação, que consiste em desconsiderar o tempo de execução das heurísticas de baixo nível durante as redefinições dos valores que indicam a qualidade desses operadores. Mais especificamente, a estratégia para essas redefinições passa a ser:

- Se uma heurística H_i for executada logo após H_{ant}, diminuindo em L_i o custo da solução (observe que L_i será negativo quando a qualidade da solução piorar), as seguintes alterações serão realizadas:

  $$f_1(H_i) = \psi_1\ f_1(H_i) + L_i$$
  $$f_2(H_i, H_{ant}) = \psi_2\ f_2(H_i, H_{ant}) + L_i$$

  Observe que, na estratégia de Soubeiga (implementada inicialmente), o valor L_i era dividido pelo tempo de CPU consumido por H_i.

4.2 Contribuições ao Algoritmo Genético Hiper-Heurístico

Na Seção 3.4, apresentamos as duas primeiras versões [13, 34] do algoritmo genético hiper-heurístico de Cowling et al. Considerando que uma qualidade crucial para esse algoritmo é a capacidade de encontrar seqüências de heurísticas capazes de influenciar positivamente a busca, notamos que urgia a incorporação da idéia da otimização local gulosa (buscar sempre o melhor vizinho da solução corrente) por esse tipo de hiper-heurística, uma vez que o caminho trilhado por um processo de otimização local gulosa que se baseie em operadores para construção de vizinhança representa justamente uma seqüência de operadores metodicamente construída para aplicação à solução inicial. Com isso em mente, desenvolvemos uma nova forma de mutação, que será apresentada na Seção 4.2.2. Também descrevemos, na Seção 4.2.3, algumas alterações adicionalmente experimentadas no algoritmo de Cowling et al.

4.2.1 Implementação da Hiper-Heurística

Para possibilitar o início dos experimentos, implementamos a hiper-heurística de Cowling et al. de acordo com a segunda versão publicada pelos autores [34] (embora certos detalhes dessa versão não constem na respectiva referência e os tenhamos obtido em outras publicações que se referem à mesma hiper-heurística [13, 32, 33]). A Figura 4.2 é uma visão geral da implementação resultante, sobre a qual tecemos alguns comentários a seguir.

- Para avaliar um cromossomo, aplica-se a seqüência de heurísticas determinada por ele à melhor solução já encontrada. A medida da aptidão de um cromossomo X será calculada como $\frac{D_X}{|X|\,T_X}$, consistindo na diminuição de custo (D_X) promovida por ele, dividida pelo produto entre o tamanho do cromossomo ($|X|$) e o tempo despendido na utilização da seqüência de heurísticas (T_X). Note que esta avaliação consiste no valor inverso àquele descrito pelos autores [34] como "penalização" (consulte a Seção 3.4). Como se pode observar, interpretamos a fórmula proposta como a única forma utilizada para avaliação dos cromossomos, embora os autores não deixem completamente claro se a fórmula inicial de avaliação [13] foi de fato descartada a partir da versão que emprega cromossomos de tamanhos diversos [34].

- Os cromossomos são avaliados em diversos pontos do algoritmo, notadamente durante a construção da população inicial e ao fim de cada geração. Um exemplo de situação que exige avaliação prévia é a mutação do passo 2(c)iiA, uma vez que se deve determinar a seqüência de heurísticas mais adequada para remoção em um cromossomo que pode ter sido criado numa operação de cruzamento.

- Desconsiderando a função de aptidão adotada inicialmente por Cowling et al., temos como conseqüência a ausência da estratégia que consistia em avaliar um cromossomo de acordo com a solução obtida por ele na última execução de sua seqüência de heurísticas. Trata-se de uma conseqüência positiva, uma vez que tal função de aptidão desconsiderava o fato de que cromossomos distintos podem ter sido avaliados partindo de soluções diferentes, o que tornava menos justas as comparações entre eles.

- Em cada execução do passo 2(b)i, a probabilidade de seleção de cada cromossomo será proporcional à sua aptidão (naturalmente, em algumas gerações pode ser necessário somar uma constante aos valores de aptidão dos cromossomos, de forma a evitar probabilidades negativas). Uma vez que Cowling et al. não especificam a forma adotada em sua hiper-heurística para construção da população intermediária, optamos pelo mapeamento

implícito de aptidão, de forma a evitar os problemas ocasionados pelo mapeamento explícito (Seção 2.3).

- Dado que os autores também não apresentam a estratégia utilizada para definir quando serão aplicados os dois diferentes tipos de mutação, implementamos uma escolha aleatória, com probabilidades definidas experimentalmente. A mutação clássica será escolhida com probabilidade $P_{MC} = 0,33$ (valor escolhido do intervalo $[0,1,\ 0,6]$). Pode-se entender o valor determinado experimentalmente para P_{MC} como uma indicação da relevância, na hiper-heurística considerada, da mutação clássica perante a mutação criada por Cowling et al.

- Na mutação clássica, cada gene será alterado com probabilidade $P_{MCG} = 0,1$ (valor escolhido experimentalmente do intervalo $[0,01,\ 0,15]$).

4.2.2 Um Novo Operador de Mutação

O algoritmo genético hiper-heurístico apresentado na Seção 3.4 conta, na segunda versão, com dois operadores de cruzamento e três operadores de mutação. Nossa principal extensão a essa hiper-heurística consiste na adição de um novo operador de mutação, baseado em um processo de otimização local que parte de uma solução S do problema original e utiliza as heurísticas de baixo nível para alcançar uma solução não inferior a S. A aplicação da nova mutação a um cromossomo X é realizada de acordo com o algoritmo exibido na Figura 4.3.

Devido à presença do novo tipo de mutação, foi necessário redefinir experimentalmente as probabilidades de utilização dos operadores de mutação. Durante esses experimentos, procuramos manter a razão ($\frac{1}{2}$) entre a probabilidade da mutação clássica e a probabilidade da mutação criada por Cowling et al. O alto valor estipulado para a probabilidade do novo operador de mutação ($0,49$, escolhido experimentalmente do intervalo $[0,1,\ 0,6]$), configura, por si só, uma importante indicação da relevância da nova mutação para a hiper-heurística.

Para evitar que um cromossomo tenha seu tamanho excessivamente aumentado com a aplicação da mutação, impusemos um limite superior para o tamanho resultante: E_{max} vezes o tamanho do cromossomo pré-mutação. O valor determinado experimentalmente para E_{max} $(1,5)$ foi escolhido do intervalo $[1,1,\ 1,9]$.

4.2.3 Idéias Adicionais

Como o valor de aptidão calculado para um cromossomo se baseia na diferença de custo alcançada por sua seqüência de heurísticas, não há absolutamente nenhuma garantia de que esse

Figura 4.2: Visão Geral da Implementação.

1. Construa a população inicial, com 30 cromossomos de tamanho 14 (valores determinados por Han e Kendall [33]), cujas heurísticas componentes devem ser selecionadas de maneira aleatória.
2. Enquanto houver tempo disponível:
   (a) Seja *NovaPopulacao* uma população sem cromossomos.
   (b) (Fase de cruzamento) Repita 10 vezes:
      i. Escolha, aleatoriamente, dois cromossomos da população atual. Sejam C_1 e C_2 cópias desses cromossomos.
      ii. Realize o cruzamento entre C_1 e C_2, de acordo com os passos a seguir, com probabilidade $0,6$ (Cowling et al. [34]):
         A. Realize o cruzamento de ponto único (Seção 2.3 – ilustração na Figura 2.2), com probabilidade $0,5$ (Han e Kendall [33] determinam que os dois tipos de cruzamento devem ser equiprováveis).
         B. Caso o cruzamento de ponto único não tenha sido realizado, efetue o cruzamento *best-best* (apresentado na Seção 3.4).
      iii. Acrescente C_1 e C_2 a *NovaPopulacao*.
   (c) (Fase de mutação) Para cada cromossomo C em *NovaPopulacao*, realize a mutação, de acordo com os passos abaixo, com probabilidade $0,1$ (Cowling et al. [34]):
      i. Aplique mutação a C com a estratégia clássica (apresentada na Seção 2.3), com probabilidade P_{MC}. Neste caso, a probabilidade de mutação de cada gene será P_{MCG}.
      ii. Caso não tenha sido realizada a mutação clássica, efetue a mutação de C como a seguir:
         A. Se o tamanho de C é estritamente maior que o tamanho médio dos cromossomos, remova dele a seqüência de cromossomos que causou maior aumento de valor objetivo durante a avaliação.
         B. Caso contrário, insira em uma posição aleatória de C a melhor seqüência de heurísticas de um cromossomo C_A (aleatoriamente escolhido).
   (d) Atualize a população atual, removendo os 20 cromossomos menos aptos (grau de elitismo proposto por Han e Kendall [33]) e acrescentando os 20 cromossomos de *NovaPopulacao*.

valor será sempre positivo. Uma conseqüência desse fato é que os cromossomos com aptidão negativa serão, devido à forma da função de aptidão (Seção 4.2.1), favorecidos caso possuam muitas heurísticas ou exijam alto consumo de tempo. Esse comportamento aparentemente insensato da função de aptidão (que, no entanto, os autores justificam de maneira lógica, afirmando que a função calcula a “melhoria por unidade de tempo”) motivou a implementação e

Figura 4.3: Aplicação do Novo Operador de Mutação a Um Cromossomo X.

1. Seja c a função objetivo e seja S_m a melhor solução já encontrada pela hiper-heurística.
2. Faça $T_X = |X|$.
3. Aplique o cromossomo X a S_m, obtendo uma nova solução S.
4. Faça $MELHOROU = 1$ e $EXCEDEU = 0$.
5. Enquanto $MELHOROU = 1$ e $EXCEDEU = 0$:
   (a) Para cada heurística H_i: aplique H_i à solução S, obtendo uma solução S_i.
   (b) Seja H_k uma heurística tal que: $c(S_k) \leq c(S_i)$, para toda heurística H_i.
   (c) Se $c(S_k) < c(S)$:
      - Faça $S = S_k$.
      - Acrescente H_k ao final do cromossomo X.
   (d) Caso contrário, faça $MELHOROU = 0$.
   (e) Se $|X| > E_{max}\, T_X$, faça $EXCEDEU = 1$.

experimentação de uma mudança natural:

- Caso a diminuição de custo obtida por um cromossomo seja positiva, a aptidão será calculada como anteriormente: $\frac{D_X}{|X|\, T_X}$ (Seção 4.2.1).
- Caso contrário, a aptidão será $D_X\, |X|\, T_X$.

Uma vez que essa modificação da função de aptidão poderia provocar variações muito grandes, em módulo, entre as aptidões dos cromossomos (afetando de maneira indesejada o mapeamento de aptidão), substituímos o valor $|X|$, na função de aptidão, por $\frac{|X|}{M_{TC}}$, sendo M_{TC} o tamanho médio dos cromossomos. Substituição análoga foi realizada para T_X.

Como veremos no Capítulo 6, uma avaliação experimental da alteração acima proposta sugere-nos que tal alteração não constitui uma melhoria. Portanto, utilizamos a versão apresentada na seção anterior (com o novo operador de mutação) como base para a última alteração, que consiste em deixar de considerar o tamanho de um cromossomo ao calcular sua aptidão. A motivação para esse experimento está no fato de que o tempo consumido pelo cromossomo já é utilizado como penalização e, portanto, não é evidente a necessidade de alterar a aptidão de um indivíduo devido a seu tamanho.

Note que o consumo de tempo requerido pela seqüência de heurísticas determinada por um cromossomo será, em geral, aproximadamente proporcional ao tamanho de tal seqüência. Sendo

assim, não seria inadequado dizer que penalizar os cromossomos de acordo com seus tamanhos é apenas uma forma indireta de penalizá-los novamente devido ao consumo de tempo. É importante determinar se essa penalização dupla (que impede de maneira eficaz o crescimento exagerado dos cromossomos, mas diminui significativamente a importância da função objetivo) é realmente adequada.

4.3 Um Algoritmo Hiper-Heurístico de Estimação de Distribuição

Nesta seção apresentamos a primeira hiper-heurística originalmente introduzida por este trabalho. Essa hiper-heurística, que batizamos como *HHED*, é um algoritmo populacional em que cada indivíduo representa uma seqüência de heurísticas de baixo nível – naturalmente, inspiramo-nos no algoritmo genético hiper-heurístico de Cowling et al. ao utilizarmos essa representação. O HHED pode ser visto como um algoritmo de estimação de distribuição (consulte a Seção 2.5 para revisar esse tipo de algoritmo), uma vez que os indivíduos são construídos de forma completamente aleatória e que as probabilidades utilizadas durante esse processo são determinadas de acordo com uma análise dos melhores indivíduos da população corrente.

Uma característica importante do HHED, que contraria os algoritmos evidenciados na Seção 2.5, é o fato de que não levamos em consideração, durante a definição das probabilidades, as posições absolutas das heurísticas nas seqüências em que essas ocorrem, uma vez que o uso dessa informação não é compatível com a suposição de Cowling et al. (Seção 3.4): uma seqüência de heurísticas que oferece bons resultados dentro de um indivíduo tem boas chances de melhorar a qualidade de outro indivíduo caso seja inserida neste último. Como ilustração, considere que o número de indivíduos selecionados em cada iteração do algoritmo de estimação de distribuição seja $n_s = 3$ e que uma dessas seleções tenha obtido as seqüências $\{H_2, H_8, H_1, H_9, H_{17}\}$, $\{H_8, H_1, H_9, H_{13}, H_4\}$ e $\{H_9, H_{17}, H_8, H_1, H_9\}$, sendo a qualidade desses indivíduos influenciada fortemente pela subseqüência $\{H_8, H_1, H_9\}$, que ocorre em uma diferente posição em cada indivíduo. Para capturar esse tipo de informação desejável, não há razão óbvia para considerarmos cada posição do indivíduo como uma variável aleatória.

A estratégia implementada no HHED para a definição das probabilidades e a construção dos indivíduos baseia-se na hiper-heurística de Soubeiga, conforme será fácil notar adiante. A Figura 4.4 ilustra de forma sucinta a implementação do HHED. Note que aplicamos novamente a idéia da otimização local direta para a extensão dos indivíduos, como outrora fizemos no desenvolvimento de um novo operador de mutação (Seção 4.2.2). Abaixo descrevemos alguns dos parâmetros empregados pelo algoritmo:

- O valor $\alpha \in (0, 1]$, utilizado nas redefinições de probabilidades, representa o peso de um valor recém-inferido em relação ao valor prévio da probabilidade. α foi definido experimentalmente como $0,2$, dentre valores no intervalo $[0,1,\ 0,9]$. É natural que um valor baixo tenha sido selecionado, pois, embora valores baixos para α aumentem a chance de que o algoritmo mantenha-se sempre longe de uma convergência, o risco de se utilizarem valores mais altos é certamente mais relevante, pois uma convergência precoce é extremamente indesejável. Uma estratégia natural seria fazer de α uma variável dinâmica, com valor crescente, de forma a forçar a convergência nas últimas etapas do processo e a aumentar a exploração nas gerações iniciais. Porém, alguns experimentos com α crescendo linearmente (em relação ao tempo) de $0,05$ a $0,35$ (ou em intervalos menores com média $0,2$) sugerem que um valor fixo é mais adequado.

- O tamanho da população (t_p) foi definido como 35, após experimentos com valores no intervalo $[10,\ 100]$. É interessante observar que esse resultado deixa a população do HHED com tamanho semelhante ao da população do algoritmo genético hiper-heurístico de Cowling et al.

- O número (n_s) de indivíduos selecionados em cada iteração para inferência de probabilidades foi definido como 11. Foram realizados experimentos com diversos valores no intervalo $[1,\ 34]$.

- Como número mínimo de heurísticas componentes de cada indivíduo (t_{min}), foi utilizado o valor 36, após experimentos com valores do intervalo $[5,\ 40]$.

- O tamanho máximo (t_{max}) de cada indivíduo foi definido como 43, valor obtido experimentalmente do intervalo $[37,\ 50]$.

Cada indivíduo é construído incrementalmente, de forma aleatória, com base nas probabilidades estipuladas pelas funções p_1, p_2, p_3 e p_4 (naturalmente, implementadas como vetores). O algoritmo da Figura 4.5 ilustra esse processo, sendo $u,v,w,z \in (0,\ 1)$, com $u < v < w < z$ e $u+v+w+z = 1$, valores reais fixos. A restrição $u < v < w < z$ se deve à maior relevância das informações sobre a qualidade das seqüências que contêm mais heurísticas, considerando-se que um dos principais objetivos da hiper-heurística é detectar seqüências que tenham impacto positivo. Esses valores foram definidos experimentalmente como $u = 0,07$, $v = 0,19$, $w = 0,31$ e $z = 0,43$, tendo sido o valor de u escolhido do intervalo $[0,01,\ 0,22]$, em experimentos nos quais mantivemos sempre a relação $z - w = w - v = v - u$. A extensão de um indivíduo por otimização local direta segue o mesmo princípio da mutação apresentada na Figura 4.3, que propusemos como extensão ao algoritmo genético hiper-heurístico.

Figura 4.4: Visão Geral da Implementação.

1. Inicialize todas as probabilidades como $\frac{1}{n_h}$, sendo n_h o número de heurísticas.
2. Construa a população inicial, com t_p indivíduos de tamanho t_{min}, e avalie todos os indivíduos.
3. Estenda todos os indivíduos com otimização local direta e avalie-os. A extensão deve ser limitada, de forma que nenhum indivíduo fique com tamanho superior a t_{max}.
4. Enquanto houver tempo disponível:
   (a) Selecione os n_s melhores indivíduos da população atual e analise-os para obter as seguintes medidas:

   s_t: a soma dos comprimentos dos indivíduos.

   O_{H_i} (para cada heurística H_i): o número de ocorrências de H_i nos indivíduos selecionados.

   $O_{H_iH_j}$ (para cada par de heurísticas (H_i, H_j)): o número de ocorrências de H_i após H_j.

   $O_{H_iH_jH_k}$: o número de ocorrências de H_i após a ocorrência da seqüência $\{H_k, H_j\}$.

   $O_{H_iH_jH_kH_l}$: o número de ocorrências de H_i após $\{H_l, H_k, H_j\}$.

   F_{H_i}: o número de ocorrências de H_i como última heurística em um indivíduo.

   $F_{H_iH_j}$: o número de ocorrências da seqüência $\{H_j, H_i\}$ como sufixo em um indivíduo.

   $F_{H_iH_jH_k}$: o número de ocorrências da seqüência $\{H_k, H_j, H_i\}$ como sufixo em um indivíduo.

   (b) Redefina as probabilidades de acordo com o procedimento abaixo (quando um denominador for nulo, o termo que inclui a fração será definido como $\alpha \frac{1}{n_h}$):

   $$p_1(H_i) = (1-\alpha)p_1(H_i) + \alpha\frac{O_{H_i}}{s_t}$$
   $$p_2(H_i,H_j) = (1-\alpha)p_2(H_i,H_j) + \alpha\frac{O_{H_iH_j}}{O_{H_j} - F_{H_j}}$$
   $$p_3(H_i,H_j,H_k) = (1-\alpha)p_3(H_i,H_j,H_k) + \alpha\frac{O_{H_iH_jH_k}}{O_{H_jH_k} - F_{H_jH_k}}$$
   $$p_4(H_i,H_j,H_k,H_l) = (1-\alpha)p_4(H_i,H_j,H_k,H_l) + \alpha\frac{O_{H_iH_jH_kH_l}}{O_{H_jH_kH_l} - F_{H_jH_kH_l}}$$

   (c) Construa uma nova população, com o mesmo procedimento utilizado nos passos 2 e 3, e substitua a população atual pela recém-criada.

Para que seja possível avaliar um indivíduo X, a seqüência de heurísticas determinada por ele será aplicada à melhor solução já encontrada, de forma que possamos medir a diminuição de custo (D_X) promovida por X. Sua avaliação será, então, calculada como $D_X \frac{T_M}{T_X}$ (caso $D_X \geq 0$) ou como $D_X \frac{T_X}{T_M}$ (caso $D_X < 0$), sendo T_X o tempo gasto na aplicação da seqüência de heurísticas

Figura 4.5: Construção de Um Indivíduo.

1. Faça $T\_ATUAL = 0$.
2. Enquanto $T\_ATUAL < t_{min}$:
   - Se $T\_ATUAL = 0$, defina a probabilidade de cada heurística H_i como $p_1(H_i)$.
   - Se $T\_ATUAL = 1$, seja H_k a heurística já existente no indivíduo e seja $r = u + v$. Defina a probabilidade de cada heurística H_i como:
   $$\frac{1}{r}(u\, p_1(H_i) + v\, p_2(H_i, H_k))$$
   - Se $T\_ATUAL = 2$, sejam H_a e H_b, respectivamente, a primeira e a segunda heurísticas do indivíduo e seja $r = u + v + w$. Defina a probabilidade de cada heurística H_i como:
   $$\frac{1}{r}(u\, p_1(H_i) + v\, p_2(H_i, H_b) + w\, p_3(H_i, H_b, H_a))$$
   - Se $T\_ATUAL > 2$, sejam H_a, H_b e H_c as três últimas heurísticas do indivíduo. Defina a probabilidade de cada heurística H_i como:
   $$u\, p_1(H_i) + v\, p_2(H_i, H_c) + w\, p_3(H_i, H_c, H_b) + z\, p_4(H_i, H_c, H_b, H_a)$$
   - Selecione uma heurística aleatoriamente, de acordo com as probabilidades que foram estipuladas em um dos quatro itens anteriores, e a acrescente ao indivíduo.
   - Faça $T\_ATUAL = T\_ATUAL + 1$.

e T_M o tempo médio utilizado pelos indivíduos em operações desse tipo. Com isso, o tempo de execução de uma seqüência de heurísticas terá sempre um efeito penalizador na avaliação do indivíduo correspondente.

Observe que, embora tenhamos nos inspirado na hiper-heurística de Soubeiga para implementar a definição das probabilidades no HHED, não há nenhum cálculo baseado no fator de diversificação daquela hiper-heurística, uma vez que um papel dessa natureza cabe à construção aleatória dos indivíduos no HHED. Note que a exploração do espaço de indivíduos nunca fica limitada, dado que nenhuma probabilidade se torna nula durante a execução do algoritmo.

4.4 Uma Hiper-Heurística de Reconexão de Caminhos

O segundo método hiper-heurístico originalmente apresentado neste trabalho foi batizado como *HHRC* e é baseado na meta-heurística de reconexão de caminhos (Seção 2.4.2) com atualização

dinâmica do conjunto de referência (Seção 2.4.3). Mais uma vez, representamos os indivíduos como seqüências de heurísticas. A Figura 4.6 apresenta a estrutura geral do HHRC. Note que o conjunto de referência inicial é construído em um processo de várias etapas, que visa à produção de um conjunto com boa qualidade e com diversidade entre seus indivíduos, conforme convém à meta-heurística de reconexão de caminhos. A Figura 4.7 ilustra a construção de $CjRef$ a partir do conjunto $PreRef$, de acordo com a estratégia apresentada na Seção 2.4.1.

Figura 4.6: Visão Geral da Implementação.

1. Construa o conjunto C_0, com T_{C_0} indivíduos de tamanho t_0, gerados aleatoriamente.
2. Estenda todos os indivíduos de C_0 através de otimização local direta.
3. Construa o conjunto $PreRef$, com os melhores T_{PR_0} indivíduos de C_0.
4. A partir de $PreRef$, construa o conjunto de referência $CjRef$, com T_{CR} indivíduos.
5. Faça $NovosIndiv = 1$.
6. Repita enquanto houver tempo disponível:
   (a) Se $NovosIndiv = 0$, reconstrua o conjunto de referência, com a estratégia indicada na Seção 2.4.3.
   (b) Produza o conjunto *NovosPares*, contendo todos os pares de indivíduos (α, β) tais que $\alpha, \beta \in CjRef$.
   (c) Faça $NovosIndiv = 0$.
   (d) Enquanto $NovosPares \neq \emptyset$:
      i. Extraia um par (I', I'') de *NovosPares*.
      ii. Aplique o método de reconexão para produzir um conjunto *Comb* de indivíduos produzidos a partir de combinações entre I' e I''.
      iii. Estenda todos os indivíduos de *Comb*, a partir de otimização local direta, de forma a produzir um novo conjunto $Comb_E$.
      iv. Para cada $I \in Comb_E$:
         A. Se I é melhor que pelo menos um indivíduo de $CjRef$:
            - Remova o pior indivíduo de $CjRef$.
            - Faça $CjRef = CjRef \cup \{I\}$.
            - Reduza o tamanho de um dos elementos de $CjRef$.
            - Faça $NovosPares = \emptyset$ e $NovosIndiv = 1$.
      v. Se $NovosIndiv = 0$ e $|PreRef| < \lceil \frac{T_{CR}}{2} \rceil$, inclua o melhor indivíduo de $Comb_E$ em $PreRef$.

O cálculo da distância entre duas seqüências de heurísticas é crucial, uma vez que é utilizado tanto na construção do conjunto de referência inicial quanto na importante fase de reconexão.

Figura 4.7: Construção do Conjunto de Referência a partir de $PreRef$.

1. Seja I_M o melhor indivíduo de $PreRef$.
2. Faça $CjRef = \{I_M\}$ e remova I_M de $PreRef$.
3. Enquando $|CjRef| < T_{CR}$:
   - Seja I_D o indivíduo de $PreRef$ cuja soma das distâncias aos indivíduos de $CjRef$ é máxima.
   - Faça $CjRef = CjRef \cup \{I_D\}$ e remova I_D de $PreRef$.

Uma visão geral da implementação desse cálculo aparece na Figura 4.8. Note que, como podemos representar cada heurística de baixo nível como um inteiro no intervalo $[1, n_h]$, sendo n_h o número de heurísticas, seria possível considerar cada seqüência de heurísticas como um ponto de um espaço euclidiano, tornando mais natural o cálculo das distâncias. É importante ressaltar que essa estratégia não faz sentido, pois sua consistência dependeria da suposição de que o número de cada heurística representa sua posição em uma reta na qual todas as heurísticas se encontram e da qual é possível inferir as distâncias entre as heurísticas. Essa suposição é, no caso geral, completamente incompatível com a realidade.

Figura 4.8: Cálculo da Distância entre Indivíduos I' e I''.

1. Sejam: $t_{max} = \max\{|I'|, |I''|\}$ e $t_{min} = \min\{|I'|, |I''|\}$.
2. Inicialize a distância D como 0.
3. Para $i = 1, 2, \ldots, t_{min}$:
   - Se a i-ésima heurística de I' é diferente da i-ésima heurística de I'', faça $D = D + 1$.
4. Faça $D = D + (t_{max} - t_{min})$.

A reconstrução do conjunto de referência será feita por meio da substituição dos $\lceil \frac{T_{CR}}{2} \rceil$ piores indivíduos do conjunto de referência por elementos obtidos do conjunto $PreRef$. Sendo assim, o conjunto $PreRef$ existirá durante toda a execução do algoritmo e terá um tamanho inicial significativamente superior a T_{CR}, de forma que as soluções excedentes possam, caso necessário, ser utilizadas em reconstruções do conjunto de referência. Ademais, o algoritmo garantirá que o tamanho de $PreRef$ seja sempre suficiente para permitir uma reconstrução do conjunto de referência: quando necessário, algum indivíduo produzido na fase de reconexão será acrescentado a

PreRef.

A conexão entre dois indivíduos se efetuará de acordo com o algoritmo ilustrado na Figura 4.9. Esse algoritmo se baseia em *passos de reconexão*, que implementamos conforme a Figura 4.10. O objetivo dos passos de reconexão, partindo de um indivíduo I' em direção a um indivíduo I'', é transformar I' em I'', para, em seguida, criar um indivíduo que seja diferente de I'', mas mantenha as características comuns entre I' e I'', evitando, ao mesmo tempo, a intensificação das semelhanças com I'. Um exemplo de reconexão é exibido na Figura 4.11.

Figura 4.9: Produção de Combinações na Reconexão entre Indivíduos I' e I''.

1. Seja $DIST$ a distância entre I' e I''.
2. Seja $NumP = \lceil \frac{DIST}{3} \rceil$.
3. Efetue $NumP$ passos de reconexão partindo de I' em direção a I''. Acrescente o resultado I_{R1} a *Comb*.
4. Efetue $NumP$ passos partindo de I_{R1} em direção a I''. Acrescente o resultado I_{R2} a *Comb*.
5. Efetue $NumP + \lfloor \frac{|I''|}{3} \rfloor$ passos partindo de I_{R2} em direção a (e passando por) I''. Acrescente o resultado I_{R3} a *Comb*.
6. Repita os três passos acima, mas partindo de I'' em direção a I' e substituindo $\lfloor \frac{|I''|}{3} \rfloor$ por $\lfloor \frac{|I'|}{3} \rfloor$.

Chamemos a atenção, novamente, para a possibilidade de representação das seqüências de heurísticas como pontos de um espaço euclidiano. Com isso em mente, poderíamos implementar um processo de reconexão baseado em combinações lineares. Note, ainda, que essa idéia poderia dar origem à implementação de uma hiper-heurística baseada na meta-heurística *particle swarm optimization*, sendo as velocidades representadas como vetores euclidianos. Porém, esse tipo de abordagem sofreria de ao menos dois males, sendo o segundo deles extremamente indesejável:

- Ao combinarmos duas ou mais seqüências de heurísticas (ou aplicarmos "velocidades" a essas seqüências), freqüentemente surgiriam valores não inteiros nas coordenadas, o que exigiria arredondamentos.
- Como discutimos anteriormente, os números inteiros que representam as heurísticas não possuem nenhum significado no que diz respeito à similaridade entre esses procedimentos. Por exemplo, ao combinarmos as seqüências $\{H_2, H_2, H_2\}$ e $\{H_4, H_4, H_4\}$, pareceria

Figura 4.10: Um Passo da Reconexão entre I' e I'', Partindo de Um Indivíduo I_R e Produzindo Um Indivíduo I_N.

1. Faça $I_N = I_R$.
2. Se o prefixo de tamanho $\min\{|I'|, |I''|\}$ de I_R é igual ao prefixo de mesmo tamanho em I'':
   (a) Se $|I_R| < |I''|$, acrescente a I_N o $(|I_R|+1)$-ésimo elemento de I''.
   (b) Se $|I_R| > |I''|$, remova a última heurística de I_N.
   (c) Se $|I_R| = |I''|$:
   - Selecione uma posição k de I_R cuja heurística seja diferente em I' e I'' (ou não exista em I'). Caso isso não seja possível, selecione uma posição qualquer k de I_R.
   - Armazene na k-ésima posição de I_N uma heurística aleatoriamente escolhida, necessariamente diferente da k-ésima heurística de I'.
3. Caso contrário, faça a $(j+1)$-ésima heurística de I_N igual à $(j+1)$-ésima heurística de I'', sendo j o tamanho do maior prefixo de I_R que coincide com um prefixo de I''.

Figura 4.11: Exemplo de Reconexão entre Indivíduos I' e I''.

I' : H04 H08 H08 H01 H10 H01 H07 H05 H08

I'' : H11 H11 H08 H07 H02 H06 H02

DIST = 8 NumP = 3

Conectando I' a I'' :

H11 H11 H08 H07 H10 H01 H07 H05 H08

H11 H11 H08 H07 H02 H06 H02 H05 H08

H11 H11 H08 H06 H05 H04 H02

Conectando I'' a I' :

H04 H08 H08 H01 H02 H06 H02

H04 H08 H08 H01 H10 H01 H07

H03 H08 H08 H01 H05 H09 H07 H07 H08

lógico obtermos $\{H_3, H_3, H_3\}$. No entanto, é de se esperar que, na prática, esta nova seqüência nem sequer possua alguma relação com as duas originais.

A redução no tamanho de um indivíduo será efetuada de forma semelhante à utilizada pelo operador de mutação que Cowling et al. (Seção 3.4) desenvolveram para esse fim. O indivíduo selecionado para ter seu tamanho reduzido será um daqueles cuja avaliação for pior, excluídos os indivíduos cujo tamanho é mínimo. Para avaliar um indivíduo I, o algoritmo abaixo será

empregado, de forma que indivíduos contendo muitas heurísticas terão maiores possibilidades de rejeição.

- Seja G o ganho de qualidade (negativo, caso o custo da solução aumente) resultante da aplicação de I à melhor solução já encontrada e seja T o tempo de CPU gasto nessa aplicação.
- Seja T_m o tempo médio utilizado na aplicação das heurísticas de cada indivíduo do conjunto de referência.
- Se $G \geq 0$, a avaliação de I é $G\frac{T_m}{T}$. Caso contrário, a avaliação é $G\frac{T}{T_m}$.

Alguns valores utilizados pelo HHRC foram definidos experimentalmente, conforme detalhamos a seguir:

- O tamanho inicial dos indivíduos (t_0) foi definido como 54, após experimentos com valores do intervalo $[4, 100]$.
- A extensão por otimização local direta deixa um indivíduo com, no máximo, 175% de seu tamanho original. Foram experimentados limites no intervalo $[1,05,\ 2]$.
- T_{PR_0} foi definido como 60, após experimentos com valores no intervalo $[30,60]$.
- O tamanho do conjunto de referência é 24. Foram experimentados diversos valores do intervalo $[3,30]$.
- O tamanho do conjunto C_0, produzido no primeiro passo do algoritmo, foi definido como $T_{C_0} = 80$. Esse valor foi definido sem experimentação, uma vez que a fase inicial do algoritmo (construção do conjunto de referência) consome tempo considerável e que não é desejável que T_{C_0} tenha valor muito próximo de T_{PR_0}.

4.5 Uma Hiper-Heurística Simples

Nesta seção descrevemos uma hiper-heurística, originalmente apresentada neste trabalho, que foi desenvolvida da maneira mais simples que pudemos conceber sem abrir mão de uma função de decisão que se baseie no histórico da busca. Como observamos na discussão preliminar desta Seção 4, o objetivo aqui adotado é entender até que ponto as hiper-heurísticas sofisticadas apresentadas anteriormente conseguem superar um método básico. Esse tipo de investigação é motivado pelo fato de que as hiper-heurísticas dispõem de pouca informação sobre o problema em foco.

A hiper-heurística apresentada nesta seção foi batizada como *HHDS* e sua implementação está detalhada na Figura 4.12. Essencialmente, cada iteração do HHDS consiste no sorteio de uma heurística de baixo nível e na aplicação dessa heurística à solução corrente. Como veremos, a probabilidade de cada heurística dependerá de seu desempenho histórico na busca. Alguns símbolos utilizados pela hiper-heurística têm seu significado descrito a seguir:

- H é o conjunto das heurísticas de baixo nível.
- c é a função de custos.
- A é a função de avaliação de heurísticas. Para cada $H_i \in H$, $A(H_i)$ será atualizada sempre que H_i for executada.
- S_c é a solução corrente. Essa variável será atualizada sempre que o algoritmo obtiver uma nova solução, além de ser alterada nos casos explicitamente indicados na Figura 4.12.
- S_M é a melhor solução conhecida pelo algoritmo. Naturalmente, essa variável será verificada e, se necessário, atualizada, sempre que o algoritmo obtiver uma nova solução.
- O valor L é empregado no passo 3c, que visa a garantir que $p(H_i) \leq L\, p(H_j), \forall H_i, H_j \in H$. L foi definido como 63, após experimentos com valores do intervalo $[2, 100]$.
- Os valores N_M e N_{M_O} são utilizados, no passo 3j, como limitantes para o número de iterações sem diminuição no custo de S_M. Esses valores foram definidos como $N_M = 105|H|$ (após experimentos no intervalo $[2|H|, 200|H|]$) e $N_{M_O} = 107$ (após experimentos no intervalo $[2, 200]$). Tendo em mente que cada iteração do HHDS, quando não melhora S_M, executa uma heurística exatamente uma vez, pode-se dizer que cada heurística terá, em média, 105 chances de encontrar uma solução melhor que S_M, antes que alguma atitude seja tomada pelo algoritmo para incentivar essa melhora.

O passo 3a tem como objetivo a obtenção de valores positivos que facilitem a determinação das probabilidades das heurísticas de acordo com suas avaliações. O valor de α determina a importância da última execução de uma heurística H_i ao atualizarmos $A(H_i)$. Esse valor foi inicialmente definido como $0,35$, após experimentos no intervalo $[0,05,\ 0,95]$. Em seguida, foram realizados experimentos com α variando linearmente em relação ao tempo (sempre com média $0,35$). Os resultados obtidos com α decrescendo de $0,65$ a $0,05$ mostraram-se superiores aos obtidos com α constante ($0,35$), bem como aos obtidos com α crescente (de $0,05$ a $0,65$) ou com α decrescendo mais lentamente (sendo que a menor variação experimentada consistiu em levar α de $0,4$ a $0,3$).

Figura 4.12: Visão Geral da Implementação.

1. Para cada $H_i \in H$:
   - Aplique H_i à solução inicial S_0, obtendo uma solução S_{H_i}.
   - Faça $A(H_i) = c(S_0) - c(S_{H_i})$.
2. Faça $S_c = S_M$, $IteracoesSemMelhora = 0$ e $OtimizacoesSemMelhora = 0$.
3. Enquanto houver tempo disponível:
   (a) Para cada $H_i \in H$, faça $A_+(H_i) = A(H_i) - \min\{0, A_{min}\}$, sendo $A_{min} = \min\{A(H_i) : H_i \in H\}$.
   (b) Seja $A_{+max} = \max\{A_+(H_i) : H_i \in H\}$.
   (c) Para cada $H_i \in H$, faça $B(H_i) = A_+(H_i) + \frac{A_{+max}}{L-1}$.
   (d) Para cada $H_i \in H$, faça $p(H_i) = \frac{B(H_i)}{B_S}$, sendo $B_S = \sum_{H_i \in H} B(H_i)$.
   (e) Faça $c_a = c(S_c)$.
   (f) Sorteie uma heurística H_S, de acordo com as probabilidades definidas por p, e a aplique a S_c (note que S_c será atualizada).
   (g) Seja $D_{H_S} = c_a - c(S_c)$ o ganho de qualidade obtido por H_S.
   (h) Faça $A(H_S) = (1-\alpha)A(H_S) + \alpha D_{H_S}$.
   (i) Se S_c tem custo inferior ao da melhor solução conhecida pelo algoritmo anteriormente:
      - Realize otimização local gulosa partindo de S_c.
      - Faça $PassosSemMelhora = 0$.
   (j) Caso contrário:
      i. Faça $PassosSemMelhora = PassosSemMelhora + 1$.
      ii. Se $PassosSemMelhora = N_M$:
         A. Faça $PassosSemMelhora = 0$.
         B. Realize otimização local gulosa partindo de S_c.
         C. Se S_M foi superada no passo acima, faça $OtimizacoesSemMelhora = 0$.
         D. Caso contrário, faça $OtimizacoesSemMelhora = OtimizacoesSemMelhora + 1$.
         E. Se $OtimizacoesSemMelhora = N_{M_O}$, faça $S_c = S_M$ e $OtimizacoesSemMelhora = 0$.

O passo 3j procura realizar operações alternativas nos casos em que a busca permanece por muito tempo sem encontrar uma solução que supere todas as anteriores. A operação mais comum, nesses casos, consistirá numa otimização local gulosa partindo da solução corrente. Porém, caso a situação persista durante muitas iterações, o algoritmo voltará a solução corrente

para o ponto em que foi encontrada a melhor solução conhecida. Cada execução da otimização local gulosa terá seu número de iterações limitado a $100\,|H|$. Para esse limitante, foram experimentados diversos valores no intervalo $[|H|,\ 100\,|H|]$.

5 *Problemas Adotados e Implementações Específicas*

Para possibilitar a avaliação das hiper-heurísticas implementadas neste trabalho, foi necessário, naturalmente, aplicá-las a problemas de otimização. Para isso, adotamos dois problemas bastante distintos, tendo em mente que o principal potencial das hiper-heurísticas está em sua robustez. Consideramos, ainda, na escolha dos problemas, a disponibilidade de instâncias publicamente acessíveis e já experimentadas por algoritmos sofisticados, de forma a tornar mais consistentes os experimentos realizados, dando maior credibilidade aos resultados obtidos. Abaixo, apresentamos brevemente os problemas explorados e as origens das instâncias utilizadas neste trabalho:

- O *problema do roteamento de veículos com restrições de capacidade* (*CVRP – capacitated vehicle routing problem* [63]) é um problema NP-difícil, estudado há mais de 40 anos, no qual é dado um conjunto de pontos e devem ser determinadas diversas rotas fechadas que satisfaçam certas restrições, além de garantirem que cada ponto dado seja percorrido por ao menos uma rota e que a soma dos custos das rotas seja mínima. Para experimentação, lidamos com as 20 instâncias propostas por Golden et al. ([29] apud Cordeau et al. [11]), que podem ser obtidas via Internet [63]. Como referencial para nossas avaliações, utilizamos os custos das melhores soluções conhecidas até o ano de 2004, segundo Cordeau et al. [11].

- O *Desafio ROADEF 2005* [62], uma competição internacional com participação aberta ao público, baseou-se em um problema de escalonamento no qual é dado um conjunto de itens a serem produzidos (em uma fábrica) e deve ser determinada uma permutação dos itens que minimize o número de violações de restrições fracas, sem violar a restrição forte. Este é um problema prático (proposto pela própria empresa interessada em sua resolução) e os resultados obtidos pelos vencedores da competição, bem como as instâncias utilizadas para avaliá-los, estão disponíveis na Internet [62]. As hiper-heurísticas lidaram com as 16 instâncias do conjunto *A*, que esteve disponível a todos os participantes do

desafio desde seu princípio.

Para cada um dos problemas considerados, foi implementada uma heurística construtiva simples, para geração da solução inicial a ser utilizada como ponto de partida pelas hiper-heurísticas. A facilidade de implementação do procedimento que produz a solução inicial é um requisito importante no estudo das hiper-heurísticas, uma vez que o maior atrativo desse tipo de método é a possibilidade de aplicação simples ao tratamento de vários problemas de otimização.

Naturalmente, a experimentação com os problemas considerados exigiu a implementação de algumas heurísticas de baixo nível. Sempre tendo em mente o objetivo de tornar os experimentos mais consistentes, de acordo com a filosofia das hiper-heurísticas, optamos pelo desenvolvimento de procedimentos facilmente implementáveis. De fato, pode-se verificar que não é prática comum a utilização de processos demorados (implementações de meta-heurísticas, por exemplo) como heurísticas de baixo nível em hiper-heurísticas. Em geral, esses operadores são procedimentos de comportamento facilmente caracterizável e se assemelham aos operadores utilizados por meta-heurísticas de busca por entornos. Essa característica garante uma certa previsibilidade quanto ao tempo de execução e aos resultados da aplicação de seqüências de heurísticas, o que em geral favorece a avaliação dos operadores pelas hiper-heurísticas. O escopo deste projeto, portanto, está limitado a esse tipo de estratégia.

A seguir, descrevemos os problemas adotados, bem como as implementações que foram realizadas neste trabalho objetivando manipular diretamente tais problemas, lidando com os aspectos da resolução não tratados pelas hiper-heurísticas.

5.1 O Problema de Roteamento de Veículos com Restrições de Capacidade

O **problema do roteamento de veículos** (*vehicle routing problem – VRP*) é um problema NP-difícil que possui grande importância, devido à sua forte relação com diversos problemas práticos. Na versão mais básica desse problema, são dados:

- Um *depósito D*.
- Um conjunto C de *clientes*.
- Um custo $c(p_i, p_j) = c(p_j, p_i) > 0$, para cada $p_i, p_j \in C \cup \{D\}$.

Chamemos de *rota* a qualquer seqüência de pontos de C. Para toda rota $R = q_1, q_2, \ldots, q_{|R|}$, seja $c_t(R) = c(D, q_1) + \{\sum_{r=1}^{|R|-1} c(q_r, q_{r+1})\} + c(q_{|R|}, D)$ o custo de R. Uma solução do VRP é um conjunto de rotas $S = \{R_1, \ldots, R_{|S|}\}$ tal que:

- Cada cliente em C está presente em uma rota de S.
- $c_c(S) = \sum_{s=1}^{|S|} c_t(R_s)$ é mínimo ($c_c(S)$ é o custo de S).

Cada rota R em uma solução do VRP pode ser vista como o percurso de um veículo que parte do depósito, passa pelos clientes de R, na ordem especificada nesta rota, e retorna para o depósito. A solução em si pode ser vista como um conjunto de percursos que, juntos, atendem a todos os clientes, sendo que todos os veículos partem do depósito e devem voltar a ele. O custo de uma rota pode ser visto como a distância percorrida pelo veículo correspondente, enquanto o custo da solução pode ser visto como a distância total percorrida para que todos os clientes sejam atendidos.

Neste trabalho, lidamos com uma das versões mais simples do VRP, chamada CVRP (*capacitated vehicle routing problem* – **problema do roteamento de veículos com restrições de capacidade**). O CVRP consiste no VRP com as seguintes características adicionais:

- É dada uma constante $Q > 0$, representando a *capacidade* dos veículos.
- A cada cliente $C_i \in C$ está associada uma *demanda* $d_i > 0$.
- Para toda rota R, a soma das demandas dos clientes de R deve ser no máximo Q.

Sendo assim, pode-se dizer que, no CVRP, a soma das demandas dos clientes atendidos por um veículo não deve exceder a capacidade deste, sendo tal capacidade a mesma para todos os veículos. Uma vez que, neste trabalho, 8 das 20 instâncias utilizadas na fase de experimentação possuem uma restrição não originalmente presente no CVRP, convém ressaltar que lidamos com uma pequena variação do problema, em que os comprimentos (ou custos) das rotas são limitados, como descrevemos a seguir:

- É dada uma constante $c_{max} > 0$.
- Para toda rota R, $c_t(R)$ deve ser no máximo c_{max}.

5.1.1 Construção da Solução Inicial

Para a construção da solução que é passada à hiper-heurística, utilizamos o clássico algoritmo de Clarke e Wright ([10], citado na Internet [63]), também chamado de *Savings*. Uma visão geral do algoritmo está apresentada na Figura 5.1 e sua idéia central é ilustrada abaixo:

- Sejam A e B duas rotas.

- Seja $a_{|A|}$ o último cliente atendido por A e seja b_1 o primeiro cliente atendido por B.
- Se um veículo parte do depósito, atende aos clientes de A, atende aos clientes de B e retorna ao depósito, dizemos que esse veículo percorreu uma rota H que é a fusão entre A e B.
- Como $c_t(H) = c_t(A) - c(a_{|A|}, D) + c(a_{|A|}, b_1) + c_t(B) - c(D, b_1)$, pode-se dizer que a economia (*saving*) de custo obtida na fusão entre A e B é $c(a_{|A|}, D) + c(D, b_1) - c(a_{|A|}, b_1)$. A economia, portanto, depende apenas do último cliente da primeira rota e do primeiro cliente da segunda rota.

Figura 5.1: Visão Geral do Algoritmo *Savings*.

1. $\forall p_i, p_j \in C$, calcule a economia $s(p_i, p_j) = c(p_i, D) + c(D, p_j) - c(p_i, p_j)$.
2. Seja S a solução trivial, em que cada rota atende a exatamente um cliente.
3. Considerando as economias positivas em ordem decrescente, execute estes passos:
   (a) Seja $s(p_i, p_j)$ a economia considerada nesta iteração.
   (b) Se existem, em S, uma rota A que termine em p_i e uma rota B que comece em p_j:
      - Construa uma rota H, concatenando B ao final de A.
      - Se H é viável, faça $S = (S - \{A, B\}) \cup \{H\}$.

Considerando-se apenas as instâncias do CVRP utilizadas neste trabalho, podemos afirmar que, embora o algoritmo *Savings* seja capaz de construir, com muita eficiência, soluções extremamente superiores à solução trivial de cada instância, a utilização desse algoritmo não se mostrou relevante para este trabalho, uma vez que disponibilizamos centenas de segundos para cada execução de hiper-heurística e que em poucos segundos uma hiper-heurística partindo da solução trivial consegue obter soluções semelhantes (no que diz respeito ao custo) às obtidas pelo método *Savings*.

5.1.2 Heurísticas Implementadas

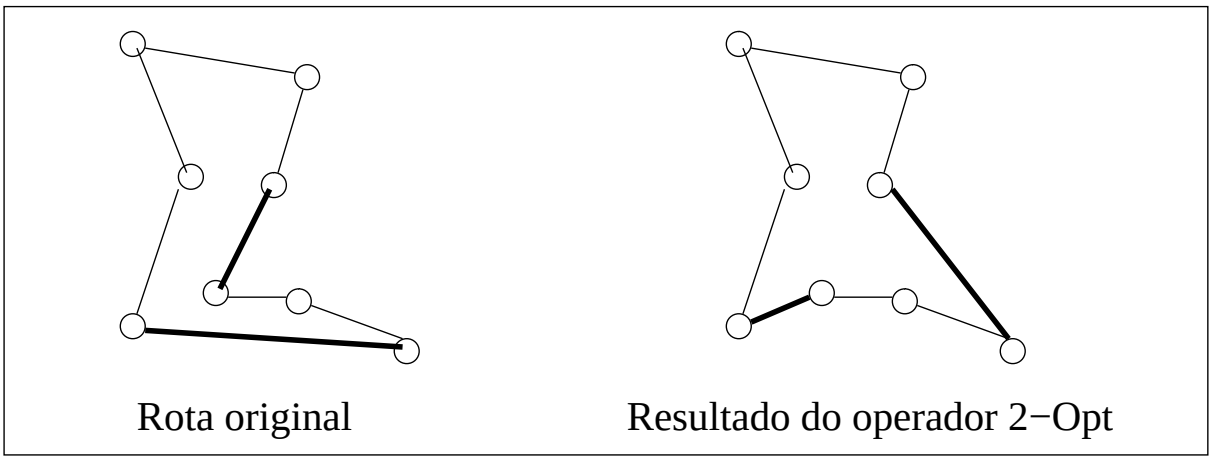
Para lidar com o CVRP, implementamos 6 operadores, sendo 4 deles baseados em procedimentos clássicos. Em alguns experimentos preliminares realizados apenas com esses 4 operadores, notamos que as hiper-heurísticas, em geral, não conseguiam melhorar as soluções após decorridos cerca de 20% do tempo de execução disponível. Essa observação motivou o desenvolvimento de 2 operadores adicionais, que apenas efetuam modificações aleatórias nas

soluções. Esses operadores possibilitaram uma maior exploração do espaço de busca, melhorando ligeiramente os resultados obtidos pelas hiper-heurísticas. Além disso, com a presença desses 2 operadores, o processo de avaliação das hiper-heurísticas é favorecido, uma vez que a escolha dos operadores a serem utilizados se torna ainda mais crítica quando há diferenças tão gritantes entre eles. A seguir, descrevemos os 6 operadores implementados.

O operador *2-Opt* ([43] apud Backer et al. [1]), desenvolvido originalmente para o problema do caxeiro viajante ([12], Seção 34.5), consiste na remoção de duas arestas de uma rota, que deve, então, ser reconstruída. Um exemplo dessa operação pode ser visto na Figura 5.2. Em nossa implementação, cada execução do operador considera todos os pares α, β em que α e β são arestas de uma mesma rota e efetua a operação utilizando o par de arestas que promove maior diminuição no custo total da solução. Apenas operações que mantêm a viabilidade da solução são consideradas pelo operador.

Dado que, na implementação realizada neste trabalho, cada solução carrega consigo algumas informações sobre cada uma de suas rotas (como o custo da rota e a soma das demandas dos clientes atendidos por ela), é importante ressaltar que a alteração no custo da rota resultante da operação realizada pelo 2-Opt pode ser calculada em tempo constante, pois depende apenas dos tamanhos das quatro arestas envolvidas (duas novas e duas eliminadas). Como conseqüência, é trivial verificar se, após a operação, a restrição de comprimento da rota continuará satisfeita (não é necessário verificar a restrição de capacidade, dado que o operador 2-Opt jamais altera o conjunto dos clientes visitados por uma rota).

Figura 5.2: Exemplo de Aplicação do Operador 2-Opt.



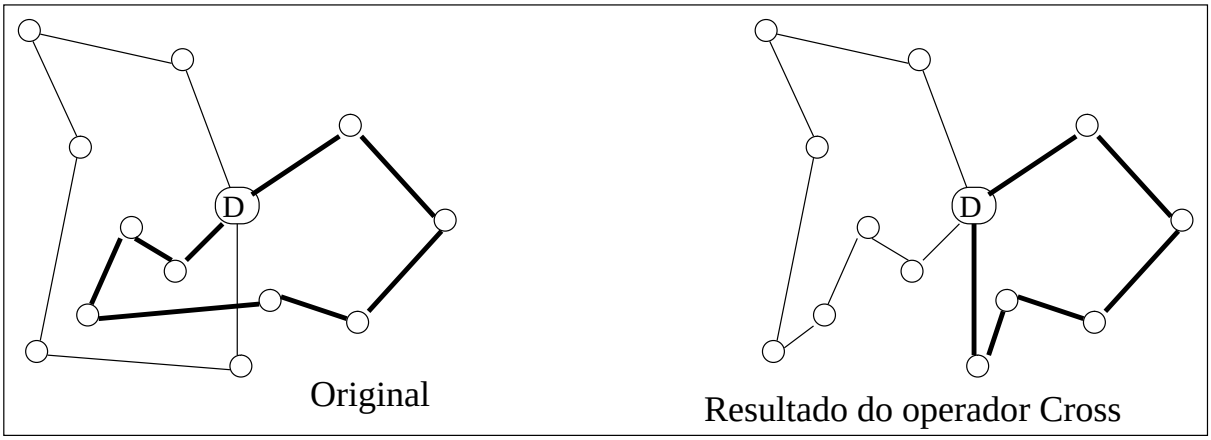
O operador *Relocate* [1] remove um cliente de uma rota e o coloca em outra rota ou em outra posição da mesma rota. Em nossa implementação, cada execução do operador analisa todos os movimentos viáveis e efetua a operação que promove maior diminuição no custo da solução.

Note que essa diminuição de custo pode ser calculada em tempo constante (para cada movimento considerado) e que também é trivial verificar se as restrições do problema continuam sendo respeitadas.

O operador *Exchange* [1] realiza um intercâmbio entre dois clientes. Esses clientes podem estar na mesma rota ou em rotas diferentes. Em nossa implementação, cada execução do operador analisa todos os intercâmbios viáveis e efetua aquele que promove maior diminuição no custo total da solução. A diminuição de custo pode ser calculada em tempo constante (para cada intercâmbio considerado) e é trivial verificar se as restrições continuam satisfeitas após a operação.

O operador *Cross* [1] troca os últimos clientes de uma rota com os últimos clientes de outra rota (sem alterar a ordem das visitas). Um exemplo dessa operação pode ser visto na Figura 5.3, em que o depósito está marcado com a letra *D*. Em nossa implementação, cada execução do operador considera todos os pares de sufixos de rotas (exceto, obviamente, para os sufixos ocorrendo em uma mesma rota) e efetua a troca que promove maior diminuição no custo da solução. Apenas operações viáveis são analisadas pelo operador.

Figura 5.3: Exemplo de Aplicação do Operador Cross.



Como expusemos anteriormente, foram implementados dois operadores que efetuam alterações aleatórias na solução considerada. Um desses operadores, que chamaremos de *Relocate aleatório*, repete cerca de n_R (o número de rotas) vezes um procedimento que consiste em sortear um cliente e movê-lo para uma posição aleatória de uma rota escolhida aleatoriamente (considerando-se também a possibilidade de criar uma nova rota para o cliente sorteado). O outro operador repete cerca de n_R vezes um procedimento que consiste em sortear dois clientes e realizar um intercâmbio entre eles. Em ambos os operadores, os sorteios realizados consideram apenas as possibilidades que mantêm a viabilidade da solução.

As soluções de boa qualidade, em geral, não permitem muitas modificações viáveis. Sendo

assim, é fácil concluir que o operador Relocate aleatório normalmente construirá ao menos uma nova rota ao ser executado com uma solução de baixo custo. Com isso, criam-se diversas possibilidades de movimentos viáveis, o que torna realmente possível uma maior exploração do espaço de busca. É crucial, portanto, que os operadores não considerem possibilidades inviáveis nos sorteios realizados, uma vez que haveria alta probabilidade de que fossem selecionados movimentos inviáveis, o que tornaria sem efeito a grande maioria dos sorteios (a não ser que os movimentos inviáveis fossem de fato executados, o que não é o caso).

5.2 O Desafio ROADEF 2005

Na versão da competição ROADEF ocorrida no ano 2005 [62], os 55 times participantes lidaram com um problema prático de escalonamento, definido por uma empresa fabricante de automóveis. Todas as instâncias disponibilizadas no desafio foram também fornecidas pela fabricante e, portanto, possuem grande relevância prática. O problema considerado, que chamaremos de *problema da permutação de veículos*, consiste na ordenação de um conjunto de carros de forma a minimizar a penalização devida à violação de restrições fracas e a satisfazer a restrição forte. A ordenação representa a ordem em que os carros serão produzidos em um dia de trabalho. A seguir, apresentamos os principais conceitos envolvidos no problema:

- **Limite de pintura**. É dado um número natural B_L, indicando o número máximo de carros de mesma cor que podem ser produzidos em seqüência. Essa é a única restrição forte do problema e a motivação para ela é o fato de que as pistolas de tinta devem ser lavadas regularmente.

- **Número de mudanças de cor**. Com o objetivo de minimizar a utilização de solvente de tinta, é desejável que o número de mudanças de cor ocorridas na seqüência de carros produzidos seja baixo. Minimizar esse número é, portanto, um dos objetivos. Cabe ressaltar que, na contagem do número de mudanças de cor, o dia de produção anterior também deve ser considerado: conta-se uma mudança adicional se a cor do primeiro carro escalonado for diferente da cor do último carro produzido no dia anterior (que é dada). Como exemplo, considere uma instância do problema em que são dados 6 carros a serem escalonados (4 verdes, 1 branco e 1 preto) e é informado que a cor do último carro do dia de trabalho anterior é azul. Para essa instância, a solução exibida na Figura 5.4 representa uma solução com 4 mudanças de cor.

- **Restrições de razão**. É desejável que os carros que requerem certa operação especial de montagem (instalação de ar condicionado, por exemplo) sejam produzidos em momentos

distantes entre si. A essas operações especiais estão associadas *restrições de razão*. Cada restrição R_R envolve determinados carros e deve estipular dois números naturais N e P, indicando que, para que R_R seja satisfeita, é necessário haver no máximo N carros associados a ela em cada seqüência de P carros que ocorra na solução. Dizemos que R_R é uma restrição de razão N/P.

- **Tipos de restrições de razão**. Cada restrição de razão é classificada como uma restrição de alta prioridade ou de baixa prioridade. Restrições de razão com baixa prioridade terão menor influência na função objetivo. Note que todas as restrições de razão são fracas. Sendo assim, todas devem ser vistas como objetivos de otimização e não como restrições invioláveis.

- **Cálculo das restrições de razão**. Para calcular o número de violações de uma restrição R_R de razão N/P, devem ser consideradas as seguintes $L+P-1$ (sendo L o número de veículos a serem escalonados) seqüências de carros:

  – As $P-1$ seqüências de P veículos que se iniciam no dia de produção anterior e terminam no dia atual. Sendo assim, serão dados pelo menos os últimos $P-1$ veículos produzidos no dia anterior.

  – As $L-P+1$ seqüências de P veículos inteiramente contidas no dia atual (a rigor, deveríamos dizer que há $\max\{0,\ L-P+1\}$ seqüências desse tipo).

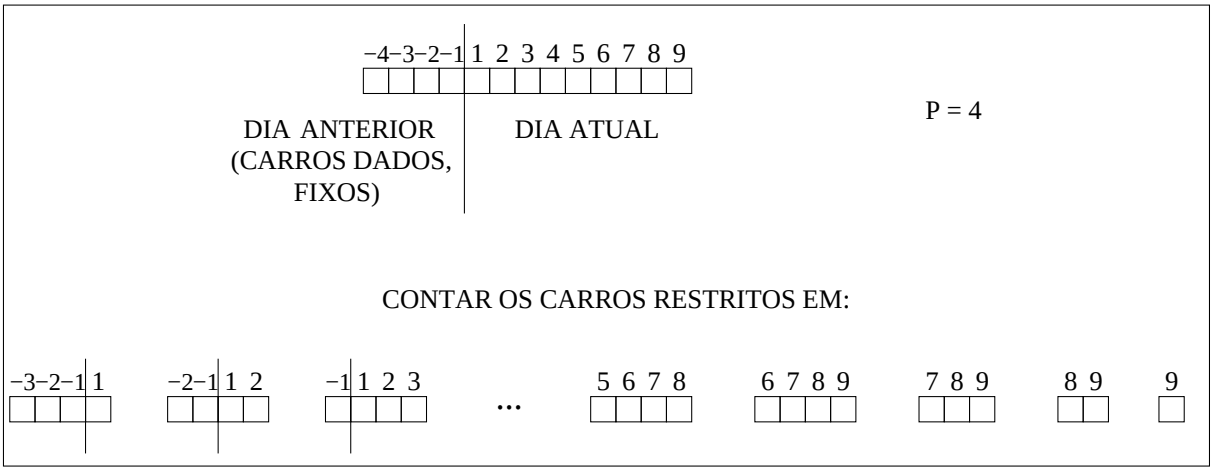
  – Os últimos $P-1$ sufixos (cada um deles com até $P-1$ veículos) da seqüência de carros produzidos no dia atual (a rigor, deveríamos dizer que temos $\min\{P-1,\ L\}$ seqüências desse tipo).

  Em cada uma dessas seqüências de veículos, deve ser observado o número X de carros envolvidos na restrição R_R. Dizemos, então, que o número de violações de R_R na seqüência considerada é $\max\{0,\ X-N\}$. O número total de violações de R_R é a soma das violações que ocorrem nas $L+P-1$ seqüências consideradas. A Figura 5.5 ilustra as seqüências que devem ser analisadas no caso de uma restrição com $P=4$, em um exemplo com $L=9$.

Como vimos acima, há três tipos de restrições desejáveis que devemos observar ao avaliarmos as soluções: número de mudanças de cor, restrições de razão de alta prioridade e restrições de razão de baixa prioridade. A importância relativa dada a cada um desses tipos de restrição fraca será estipulada em cada instância. Portanto, a função objetivo utilizada em uma instância pode diferir significativamente daquela utilizada em outra instância. Mais especificamente, como parte dos dados da instância, será dada apenas uma das 5 ordenações abaixo:

Figura 5.4: Exemplo de Solução com 4 Mudanças de Cor, em Uma Instância com 6 Carros.

Figura 5.5: Seqüências Analisadas para Restrição de Razão N/4, em Instância com 9 Carros.



- **Restrições de razão de alta prioridade; restrições de razão de baixa prioridade; número de mudanças de cor**. Neste caso, o valor objetivo, a ser minimizado, será $10^6 V_A + 10^3 V_B + N_M$, sendo V_A o número total de violações de restrições de razão de alta prioridade, V_B o número total de violações de restrições de razão de baixa prioridade e N_M o número de mudanças de cor na solução.

- **Restrições de razão de alta prioridade; número de mudanças de cor; restrições de razão de baixa prioridade**. O objetivo é minimizar $10^6 V_A + 10^3 N_M + V_B$.

- **Restrições de razão de alta prioridade; número de mudanças de cor**. O objetivo é minimizar $10^6 V_A + 10^3 N_M$.

- **Número de mudanças de cor; restrições de razão de alta prioridade; restrições de razão de baixa prioridade**. O objetivo é minimizar $10^6 N_M + 10^3 V_A + V_B$.

- **Número de mudanças de cor; restrições de razão de alta prioridade**. O objetivo é minimizar $10^6 N_M + 10^3 V_A$.

5.2.1 Construção da Solução Inicial

Em cada instância fornecida pelo Desafio ROADEF 2005, a ordem em que os carros são dados já configura uma solução obtida por uma heurística. Isso se deve ao fato de que a própria empresa que provê as instâncias possui uma implementação da meta-heurística de recozimento simulado para a resolução do problema da permutação de veículos. Portanto, embora tenhamos desenvolvido heurísticas construtivas para a geração de uma solução inicial a ser fornecida às hiper-heurísticas, empregamos efetivamente como solução inicial a melhor solução entre a fornecida na competição e a produzida por nossas heurísticas construtivas.

Como descrevemos anteriormente, cada instância no Desafio ROADEF 2005 é acompanhada por uma especificação da função objetivo a ser utilizada. A principal distinção, bastante importante, entre os 5 possíveis tipos de função objetivo é o fato de que 2 deles consideram o número de mudanças de cor como principal critério de avaliação, ao passo que os outros 3 dão maior importância ao número de violações das restrições de razão de alta prioridade. Com isso em mente, desenvolvemos duas diferentes heurísticas construtivas: uma heurística que visa à construção de uma solução com número mínimo de mudanças de cor e uma heurística que constrói uma solução procurando violar poucas restrições de razão de alta prioridade. Naturalmente, cada instância contará com apenas uma heurística construtiva para geração da solução inicial, de acordo com o critério a que sua função objetivo dá maior relevância.

A Figura 5.6 apresenta brevemente a heurística utilizada para construção da solução inicial quando o primeiro objetivo é minimizar o número de violações das restrições de razão de alta prioridade. A heurística estabelece um critério simples para tentar prever quais restrições tendem a ser violadas com maior intensidade, pois a estratégia consiste em escalonar primeiramente os carros envolvidos nessas restrições. Além disso, a heurística procura manter um espaçamento uniforme, no escalonamento, entre os carros envolvidos em uma mesma restrição. Não discutiremos essa heurística em detalhes, pois ela não foi importante para este trabalho. De fato, em raros casos ela foi capaz de produzir uma solução superior à fornecida pela própria instância considerada (e, portanto, poucas vezes a solução gerada pela heurística construtiva foi efetivamente utilizada).

É importante ressaltar que nosso critério para escolha da solução inicial (utilizar a melhor das duas soluções em mãos – a solução fornecida na instância e a solução gerada por uma de nossas heurísticas construtivas) não é uma estratégia importante do algoritmo, uma vez que a qualidade da solução final encontrada por uma hiper-heurística dependerá, em geral, muito mais das características da solução inicial que de seu custo. No entanto, dificilmente outro critério de escolha igualmente eficiente poderia ser apontado como superior ao aqui utilizado.

Figura 5.6: Heurística Construtiva Focando o Número de Violações das Restrições de Razão de Alta Prioridade.

1. Seja L o número total de veículos a serem escalonados.
2. Para cada restrição R de razão N/P, sejam:
   (a) $V_r(R)$ número de veículos envolvidos em R.
   (b) $D(R) = V_r(R) - L\,\frac{N}{P}$.
3. Ordene as restrições de razão de maneira que as restrições de alta prioridade apareçam antes das restrições de baixa prioridade. Como segundo critério da ordenação, as restrições devem estar organizadas em ordem decrescente, de acordo com a função D definida no item anterior.
4. Começando com um escalonamento vazio e seguindo a ordenação de restrições estabelecida acima, execute os seguintes passos para cada restrição R:
   (a) Seja I_L a primeira posição livre no escalonamento e F_L a última.
   (b) Seja $V_{NE}(R)$ o conjunto dos veículos envolvidos na restrição R que ainda não foram escalonados.
   (c) Faça $N_E = |V_{NE}(R)|$.
   (d) Escalone todos os veículos de $V_{NE}(R)$, começando pela posição I_L e procurando sempre fazer com que cada veículo escalonado esteja cerca de $\frac{F_L - I_L}{N_E - 1}$ posições à frente do escalonado anteriormente.

A Figura 5.7 descreve a heurística empregada na construção da solução inicial quando o primeiro critério de otimização é o número de mudanças de cor. A seguir, apresentamos algumas definições necessárias para a compreensão do algoritmo e para as análises a serem realizadas:

- C é o conjunto das cores.
- c_{DA} é a cor do último carro produzido no dia anterior.
- $V_c(c_i)$ é o conjunto dos veículos de cor c_i presentes na instância (desconsiderando-se o dia anterior), $\forall c_i \in C$.
- $V_{NE}(c_i)$ é o conjunto dos veículos que estão em $V_c(c_i)$ e ainda não foram escalonados, $\forall c_i \in C$.
- $M_s(c_i) = \lceil \frac{|V_c(c_i)|}{B_L} \rceil$, $\forall c_i \in C$. Note que B_L é o limite de pintura, definido anteriormente.
- N_M é o número total de mudanças de cor no escalonamento, conforme calculado pela função objetivo.

- N_{MD} é o número de mudanças de cor no escalonamento, desconsiderando-se a cor do último carro do dia anterior. Em outras palavras: o número de mudanças de cor existentes entre carros escalonados no dia considerado pela instância.
- $M_{N_{MD}} = (\sum_{c_i \in C} M_s(c_i)) - 1$.
- Para toda cor c, chamaremos de *seqüência maximal de carros de cor c* a qualquer seqüência de carros de cor c, escalonados de forma adjacente, que não esteja contida em outra seqüência com as mesmas características.

Figura 5.7: Heurística Construtiva Visando à Minimização do Número de Mudanças de Cor.

1. O escalonamento será iniciado a partir de uma solução parcial vazia.
2. Se $V_{NE}(c_{DA}) \neq \emptyset$:
   - Escalone $\min\{|V_{NE}(c_{DA})|, B_L\}$ carros de cor c_{DA}.
   - Execute o algoritmo da Figura 5.8.
   - Se o algoritmo da Figura 5.8 encontrou uma solução, utilize-a como resultado (neste caso, o escalonamento está concluído).
3. Execute o algoritmo da Figura 5.8, começando com uma solução parcial vazia. Se for encontrada uma solução, utilize-a como resultado. Caso contrário, não foi possível construir uma solução viável.

Figura 5.8: Núcleo do Algoritmo para Minimização do Número de Mudanças de Cor.

1. Se não houver carros escalonados:
   (a) Seja c_m uma cor tal que $|V_{NE}(c_m)| \geq |V_{NE}(c)| \ \forall c \in C$.
   (b) Escalone $\min\{|V_{NE}(c_m)|, B_L\}$ carros de cor c_m.
2. Enquanto houver carros não escalonados:
   (a) Seja c_a a cor do último carro que foi escalonado.
   (b) Se não há carros a escalonar com cor diferente de c_a, termine a execução deste algoritmo (neste caso, não foi possível construir uma solução viável).
   (c) Seja c_m uma cor tal que $c_m \neq c_a$ e $|V_{NE}(c_m)| \geq |V_{NE}(c)| \ \forall c \in C - \{c_a\}$.
   (d) Escalone $\min\{|V_{NE}(c_m)|, B_L\}$ carros de cor c_m.

Para todas as instâncias disponibilizadas no Desafio ROADEF 2005 que têm como primeiro critério o número de mudanças de cor (um total de 22 instâncias, sendo 4 delas pertencentes ao

conjunto A da competição e, portanto, utilizadas neste trabalho), a heurística construtiva descrita na Figura 5.7 constrói soluções viáveis com número mínimo de mudanças de cor. Esse comportamento é a principal causa do fato de que, na grande maioria das instâncias disponibilizadas no Desafio ROADEF 2005, a solução produzida por nossa heurística construtiva é superior à solução fornecida na instância considerada. Esse fato motivou, ainda, a implementação de todos os operadores de maneira a garantir que o número de mudanças de cor jamais será alterado pela hiper-heurística, conforme veremos na Seção 5.2.2.

Mais adiante, ficará claro por que a heurística construtiva descrita na Figura 5.7 constrói uma solução com número mínimo de mudanças de cor (caso construa uma solução viável) e por que, na prática, a solução construída tende a ser viável. Os resultados a seguir serão de grande valor em nossas análises:

- $N_{MD} \leq N_M \leq N_{MD} + 1$, claramente.
- $\forall c_i \in C$, $M_s(c_i)$ é o número mínimo de seqüências maximais de carros de cor c_i que devem aparecer na solução, uma vez que o limite de pintura deve ser respeitado.
- Como conseqüência do item anterior, temos que $N_{MD} \geq M_{N_{MD}}$.
- Caso o algoritmo da Figura 5.7 construa uma solução viável, essa solução possui N_{MD} mínimo, uma vez que, para cada cor c_i, são escalonadas exatamente $M_s(c_i)$ seqüências de cor c_i e, portanto, $N_{MD} = M_{N_{MD}}$.

A seguir, demonstramos as minimizações realizadas pela heurística construtiva da Figura 5.7 e apresentamos conclusões importantes a serem observadas nos casos em que a heurística não constrói uma solução viável.

1. Caso seja construída uma solução viável na primeira tentativa (item 2, na Figura 5.7), tal solução minimiza N_M.
    - Sabemos que o algoritmo minimiza N_{MD}. Como a cor do primeiro carro escalonado é igual à cor do último carro produzido no dia anterior, temos que $N_M = N_{MD}$ e, portanto, N_M é mínimo.
2. Caso a primeira tentativa falhe, não existe solução tal que $N_M \leq M_{N_{MD}}$.
    (a) Note que, se a primeira tentativa falhou, houve um momento (passo 2b da Figura 5.8) em que todos os carros não escalonados tinham cor c_f e esta era justamente a cor da última seqüência de carros escalonados.

(b) Portanto, o escalonamento parcial construído é composto por seqüências maximais de carros de cor c_f intercaladas com seqüências maximais de carros de outras cores. Mais formalmente, podemos dizer que o escalonamento se inicia com 0 ou 1 seqüência maximal de carros de cor c_f, seguida por 0 ou mais pares de seqüências maximais em que a primeira tem uma cor qualquer, diferente de c_f, e a segunda tem cor c_f. Essa afirmação pode ser compreendida com o auxílio do raciocínio abaixo:

i. Seja s_e o número de seqüências maximais de carros escalonados. Sabemos que a última dessas seqüências possui cor c_f.

ii. A $(s_e - 1)$-ésima seqüência escalonada (caso exista) tem uma cor $c_p \neq c_f$.

iii. A $(s_e - 2)$-ésima seqüência escalonada (caso exista) deve ter cor c_f, pois:

- No momento em que a seqüência foi escalonada, c_f era a cor mais freqüente entre os carros não escalonados (caso contrário, o algoritmo não teria falhado ao tentar escalonar outra seqüência após a última).
- A $(s_e - 3)$-ésima seqüência escalonada (caso exista) não pode ter cor c_f, uma vez que, se tivesse, a $(s_e - 1)$-ésima seqüência também teria cor c_f (por ser esta a cor mais freqüente no momento em que foi escalonada a $(s_e - 1)$-ésima seqüência).

(c) Com isso, pode-se concluir que é impossível escalonar todos os carros de cor c_f sem violar o limite de pintura, caso os demais carros sejam escalonados em um número mínimo de seqüências maximais e a primeira cor utilizada seja a cor do último carro produzido no dia anterior.

3. Note que, para que a primeira tentativa falhe, é necessário termos uma cor c_f tal que $M_s(c_f) > (\sum_{c_i \in C - \{c_f\}} M_s(c_i)) + a_{c_f}$, sendo $a_{c_f} = \begin{cases} 1 & \text{se } c_f = c_{DA} \\ 0 & \text{caso contrário} \end{cases}$.
De maneira imprecisa, podemos dizer que, para o algoritmo ser mal-sucedido em sua primeira tentativa, deve haver uma cor mais freqüente que todas as outras cores juntas. Esse fato mostra que a ausência de falhas do algoritmo em nossos experimentos é um resultado natural.

4. Se a segunda tentativa de escalonamento (item 3 da Figura 5.7) for bem-sucedida, a solução construída minimiza N_M.

- Dado que a primeira tentativa falhou, sabemos (pois vimos acima) que não existe solução com $N_M \leq M_{N_{MD}}$.
- Uma vez que a solução construída na segunda tentativa utiliza o número mínimo de

seqüências maximais de carros de cada cor, sabemos que ela minimiza N_{MD} e que, portanto, $N_M = M_{N_{MD}} + 1$.

- Como conseqüência dos dois itens acima, o número total de mudanças de cor é mínimo.

5. Se a segunda tentativa de escalonamento não construir uma solução viável e $c_f \neq c_{DA}$, não existe solução com $N_M \leq M_{N_{MD}} + 1$.

   (a) Com raciocínio similar ao utilizado na demonstração do item 2b, pode-se concluir que é impossível escalonar todos os carros de cor c_f caso os demais carros sejam escalonados em um número mínimo de seqüências maximais. Portanto, $M_s(c_f) > (\sum_{c_i \in C - \{c_f\}} M_s(c_i)) + 1$ e não existe solução com $N_{MD} = M_{N_{MD}}$.

   (b) Como $c_f \neq c_{DA}$, qualquer solução viável que seja construída começando com um carro de cor c_f terá pelo menos $M_{N_{MD}} + 2$ mudanças de cor no total.

   (c) Qualquer solução viável que seja construída começando com um carro cuja cor é diferente de c_f terá $N_{MD} \geq M_{N_{MD}} + 2$, pois:

   - Como é necessário escalonar todos os carros de cor c_f, devemos ter ao menos $2\,M_s(c_f)$ seqüências maximais no total.
   - Sendo assim, teremos $N_{MD} \geq 2\,M_s(c_f) - 1 = M_s(c_f) + M_s(c_f) - 1$.
   - Com isso, pelo item 5a, temos $N_{MD} > M_s(c_f) + (\sum_{c_i \in C - \{c_f\}} M_s(c_i)) + 1 - 1 = \sum_{c_i \in C} M_s(c_i) = M_{N_{MD}} + 1$.

O algoritmo da Figura 5.7 poderia ser estendido para produzir sempre uma solução viável com número mínimo de mudanças de cor (caso a instância seja viável). Porém, uma vez que o algoritmo implementado realiza essa tarefa para todas as instâncias experimentadas, uma extensão desse tipo não modificaria os resultados de nossos experimentos e, por isso, não foi implementada.

5.2.2 Heurísticas Implementadas

No que diz respeito aos operadores implementados para lidar diretamente com o problema da permutação de veículos, novamente adotamos tratamentos diferenciados para as instâncias, de acordo com o primeiro critério considerado pela função objetivo em cada uma delas. Foram implementados 4 operadores a serem utilizados pela hiper-heurística apenas nos casos em que o primeiro critério para otimização é o número de mudanças de cor. Para lidar com os demais casos, implementamos outras 4 heurísticas de baixo nível.

A operação básica realizada por todas as heurísticas de baixo nível desenvolvidas é a troca de posição, no escalonamento, entre duas seqüências de veículos. De fato, essa operação é a principal responsável pelo consumo de tempo das hiper-heurísticas aplicadas ao Desafio ROADEF 2005, uma vez que cada operador contém em seu cerne um laço que executa uma troca de seqüências em cada iteração. Com isso em mente, desenvolvemos, além da função de avaliação de soluções, uma *função de reavaliação*, a ser utilizada sempre que uma solução S_t for gerada a partir de uma troca realizada em uma solução S_a. Nesses casos, a função de reavaliação calcula o custo de S_t utilizando as seguintes informações:

1. Os valores obtidos durante a avaliação da solução S_a e utilizados na determinação de seu custo.

2. As posições inicial e final de cada uma das duas seqüências de veículos que foram trocadas em S_a para dar origem a S_t.

3. A própria solução S_t.

As informações descritas acima no item 1, bem como algumas informações adicionais, são armazenadas na forma de uma estrutura em linguagem C. O tipo associado a essa estrutura foi batizado como *info_sol* e cada elemento que as hiper-heurísticas enxergam como solução contém uma estrutura desse tipo (desconhecida pelas hiper-heurísticas, obviamente). Todas as estruturas do tipo *info_sol* são construídas durante os processos de avaliação de suas respectivas soluções, sem aumentar, no entanto, o consumo de tempo assintótico do cálculo do valor objetivo. Essas estruturas contêm informações como:

- O número de violações de R_R na seqüência de veículos que se inicia na posição i do escalonamento e tem comprimento P, para toda restrição R_R de razão N/P e toda posição i (incluindo as posições no dia anterior ao considerado no escalonamento).

- O número total de violações de restrições de razão de alta prioridade e o número total de violações de restrições de razão de baixa prioridade.

- O número total de mudanças de cor.

Alguns experimentos simples, realizados com instâncias providas pelo Desafio ROADEF 2005, apontaram fortes evidências de que o tempo de execução de cada operador diminui cerca de uma centena de vezes com a substituição, onde possível, da função de avaliação pela função de reavaliação.

Antes de apresentarmos as heurísticas de baixo nível desenvolvidas, é importante ressaltar que esses operadores não garantem a ausência de soluções inviáveis, especialmente durante a geração das diversas soluções que cada operador analisa antes de escolher aquela que será seu resultado. As funções de avaliação e reavaliação penalizarão as soluções inviáveis que surgirem, somando ao seu custo o valor $10^{10}\, n_I$, com $n_I = \sum_{s \in S_m} \lceil \frac{|s|}{B_L} - 1 \rceil$, sendo S_m o conjunto das seqüências maximais de veículos de mesma cor que ocorrem na solução (note que, como esperado, $n_I > 0$ se, e somente se, a solução for inviável).

Prioridade para o Número de Mudanças de Cor

Desenvolvemos 4 operadores que são passados à hiper-heurística nos casos em que o primeiro critério para otimização é o número de mudanças de cor. Todos esses operadores manterão inalterado o número de mudanças de cor que ocorrem na solução. A motivação óbvia para isso é o fato de que o algoritmo de construção da solução inicial gera soluções com número mínimo de mudanças de cor, para todas as instâncias disponibilizadas no Desafio ROADEF 2005.

A primeira heurística de baixo nível implementada, que chamaremos de h_{1p}, é bastante simples, funcionando da seguinte forma:

1. Sorteie uma posição p_1 no escalonamento. Seja v_1 o veículo presente nessa posição.
2. Para cada veículo v_i tal que v_1 e v_i têm a mesma cor e $v_1 \neq v_i$:
   (a) Troque v_1 de posição com v_i.
   (b) Reavalie a solução.
   (c) Desfaça a troca entre v_1 e v_i.
3. Troque v_1 de posição com o veículo v_m que tenha oferecido o melhor resultado no passo 2b.

A segunda heurística de baixo nível (h_{2p}) assemelha-se à anterior. Porém, o passo 1 sorteará uma seqüência maximal b_1 de veículos de mesma cor. Naturalmente, o laço que lidava com um veículo v_i em cada iteração deve agora considerar uma seqüência maximal b_i de carros de mesma cor em cada um de seus passos. No entanto, b_i não precisa ter a mesma cor de b_1. A ausência dessa última restrição pode resultar no aparecimento de algumas soluções inviáveis, mas possibilita uma maior exploração do espaço de busca, sem alterações no número de mudanças de cor presentes na solução (para garantir esta última característica, evitamos trocar de posição a primeira seqüência do escalonamento, caso seus carros tenham a mesma cor do último carro do dia anterior).

A terceira heurística de baixo nível procura explicitamente diminuir o número de violações de uma restrição de razão. Mais especificamente, a operação consiste em escolher aleatoriamente uma restrição R_R de razão N/P, violada e com alta prioridade, e uma seqüência de P veículos na qual R_R seja violada. O $(N+1)$-ésimo veículo envolvido em R_R na seqüência escolhida será trocado de posição com o veículo de mesma cor que oferecer o melhor resultado (considerando-se todos os veículos do escalonamento). Note que esta heurística é bastante similar a h_{1p}, diferindo apenas na escolha mais metódica do primeiro veículo.

O último operador implementado para as instâncias que dão prioridade ao número de mudanças de cor funciona de maneira parecida com h_{2p}. Este último operador também sorteará uma seqüência maximal b_1, mas, no passo seguinte, levará em conta apenas as seqüências que possuem a mesma cor de b_1. Uma vez escolhidas as seqüências b_1 e b_m a serem trocadas de posição, o operador não necessariamente as trocará inteiramente. Serão analisadas todas as possibilidades de trocas entre um prefixo de b_1 e o prefixo de mesmo tamanho em b_2. Por fim, será efetivamente realizada a troca que oferecer os melhores resultados.

Prioridade para as Restrições de Razão

Foram desenvolvidos 4 operadores a serem fornecidos à hiper-heurística nos casos em que o primeiro critério para otimização é o número de violações das restrições de razão de alta prioridade. A seguir, descrevemos essas heurísticas de baixo nível.

O primeiro operador implementado, que chamaremos de h_{1r}, realiza uma operação simples: sorteia uma posição p_1 no escalonamento e um número inteiro $t_s > 0$, para em seguida trocar a seqüência de veículos que se inicia em p_1 e tem tamanho t_s com a seqüência de tamanho t_s que oferecer o melhor resultado. As duas seqüências a serem trocadas de posição devem ser disjuntas. O número t_s é limitado superiormente por um valor t_m, que decresce à medida em que a execução do algoritmo de resolução se aproxima do fim.

Com as mudanças no valor da variável t_m utilizada por h_{1r}, o operador se torna mais conservador nas fases finais da hiper-heurística utilizada, favorecendo a convergência (é importante notar que essa característica do operador está totalmente definida pelo algoritmo usuário da hiper-heurística e que, portanto, a separação entre a hiper-heurística e o nível mais baixo está mantida). O intervalo estabelecido para t_m foi determinado experimentalmente. Seu limite superior está próximo de $\frac{L}{35}$, sendo L o número de veículos escalonados. A variável t_m é limitada inferiormente por um valor aproximadamente igual a $\frac{L}{235}$.

O segundo operador desenvolvido (h_{2r}) é dividido em duas fases. Primeiramente, o operador sorteia n_p pares (p_i, p_j) de posições, com $p_i \neq p_j$, e seleciona o par (p_1, p_2) que oferece o

melhor resultado caso troquemos o veículo em p_1 com o veículo em p_2. Em seguida, o operador realizará a troca entre a seqüência s_1 e a seqüência s_2 que oferecerem os melhores resultados em uma análise de todas as trocas nas quais as seqüências envolvidas (s_i e s_j) satisfazem às seguintes regras:

- s_i deve conter p_1 e s_j deve conter p_2.
- $1 \leq |s_i| = |s_j| \leq t_l$.
- s_i e s_j devem ser disjuntas.
- O veículo em p_1 deve ser trocado com o veículo em p_2.
- Se $|s_i|$ é ímpar, p_1 deve estar na posição central de s_i. Caso contrário, p_1 deve estar em uma das duas posições centrais de s_i.

Na prática, os valores que utilizamos dentro da heurística h_{2r} são $t_l = \frac{L}{100} + 10$ e $n_p = \frac{L}{t_l} + 10$. Esses valores foram determinados com alguns experimentos simples. No entanto, partimos do princípio de que t_l deve ser aproximadamente uma fração de L, uma vez que um t_l constante resultaria em trocas cujo impacto seria, em geral, mais intenso nas instâncias de menor porte.

A terceira heurística de baixo nível desenvolvida, que chamaremos de h_{3r}, procura explicitamente aumentar o tamanho de uma seqüência maximal de carros de mesma cor, considerando que isso pode resultar na diminuição do número de mudanças de cor. O primeiro passo do operador consiste em selecionar aleatoriamente uma seqüência maximal s_1 de veículos de mesma cor cujo tamanho seja estritamente menor que B_L. Em seguida, efetua-se uma troca entre uma seqüência de tamanho t_s, iniciada pelo veículo que aparece logo após s_1, e a seqüência de mesmo tamanho que oferecer o melhor resultado. O tamanho t_s será 1 caso o próximo critério a ser otimizado na solução seja o número de mudanças de cor. Dizemos que uma solução tem essa característica quando o número total de violações de restrições de razão de alta prioridade é zero e ao menos uma das seguintes situações ocorre:

- O segundo critério considerado pela função objetivo é o número de mudanças de cor.
- O número de violações de restrições de razão de baixa prioridade é zero.

Se o número de mudanças de cor não o for próximo critério a ser otimizado, o tamanho t_s será determinado aleatoriamente, utilizando o mesmo processo que descrevemos na apresentação de h_{1r}.

O último operador implementado estabelecerá como foco, em geral, um restrição de razão que esteja sendo violada. Seu funcionamento pode ser descrito da seguinte forma:

- Se o próximo critério a ser otimizado é o número de mudanças de cor, apenas execute o operador h_{3r}.
- Seja R_R uma restrição de razão N/P, selecionada aleatoriamente, que esteja sendo violada. R_R deve ser de alta prioridade, se possível.
- Selecione aleatoriamente uma seqüência de tamanho P em que R_R seja violada.
- Seja v_1 o $(N+1)$-ésimo veículo envolvido em R_R que aparece na seqüência selecionada e seja v_2 o último veículo com tais características. Sejam p_1 e p_2 as posições de v_1 e de v_2, respectivamente.
- Seja t_s um número inteiro aleatoriamente escolhido, com $0 < t_s \leq p_2 - p_1 + 1$.
- Troque a seqüência que se inicia em p_1 e tem tamanho t_s com a seqüência que oferecer os melhores resultados, dentre todas as seqüências com tamanho t_s. As seqüências trocadas devem ser disjuntas.

6 *Experimentos e Resultados*

Com a finalidade de avaliar todas as hiper-heurísticas desenvolvidas, em suas diferentes versões, aplicamo-nas a dois diferentes problemas de otimização, utilizando um total de 36 instâncias, conforme apontamos no Capítulo 5. Todos os experimentos foram realizados em um microcomputador pessoal, equipado com um processador Intel® Pentium® 4 de $1,8$ GHz e com 512 MB de memória RAM. É importante notar que, após o início dos experimentos, absolutamente nenhuma modificação foi efetuada nos algoritmos considerados, uma vez que uma ação desse tipo diminuiria a legitimidade das avaliações efetuadas com base nos resultados aqui obtidos. Em particular, alterar manualmente uma hiper-heurística ao aplicá-la a um novo problema pode inviabilizar conclusões que se refiram à robustez do algoritmo.

O procedimento que utilizamos para avaliar as hiper-heurísticas se baseia fortemente nos critérios adotados no Desafio ROADEF 2005. Uma das facetas dessa escolha foi o critério de parada definido para todas as hiper-heurísticas implementadas: cada execução de uma hiper-heurística, quando aplicada a uma instância, deve terminar antes que o intervalo de tempo entre o início e o fim do processo atinja 9 minutos. Esse tempo de execução nos permite comparar diretamente os custos das soluções obtidas neste trabalho aos custos alcançados pelos participantes do Desafio ROADEF 2005, uma vez que o tempo de execução dedicado a cada instância durante a competição (10 minutos), aliado ao microcomputador utilizado (que contava com um processador Intel® Pentium® 4 de $1,6$ GHz e com 1 GB de memória RAM), compõe um cenário com condições bastante similares às estipuladas aqui.

Todas as hiper-heurísticas foram implementadas com a capacidade de receber como argumento o tempo máximo de execução. Dessa forma, o tempo efetivamente disponibilizado não está presente no código implementado, o que reforça o fato de as hiper-heurísticas poderem ser reutilizadas sem modificações. Seria possível estabelecer um processo de avaliação mais justo, no que diz respeito à comparação entre as hiper-heurísticas implementadas, caso o tempo máximo estipulado para execução se referisse ao tempo efetivamente utilizado pelo processo no sistema operacional, direta ou indiretamente (valor que pode ser obtido, na linguagem C, através da função *clock*, encontrada na biblioteca *time*). Porém, esse tipo de critério de parada tornaria

menos intuitivas as comparações com os resultados obtidos no Desafio ROADEF 2005, além de ser menos útil do ponto de vista prático, por eliminar a possibilidade de prever com precisão o momento de término do processo.

A experimentação realizada para cada hiper-heurística consistiu em aplicar o algoritmo 5 vezes para cada instância e observar o custo médio das 5 soluções finais obtidas nessas execuções. A necessidade de aplicar a hiper-heurística múltiplas vezes a cada instância é bastante óbvia, uma vez que os resultados não são determinísticos. Contribuem para esta última característica tanto o uso de aleatoriedade quanto o fato de que o número de passos que um algoritmo consegue efetuar no tempo total de que dispõe é imprevisível, mesmo que o computador e o sistema operacional utilizados sejam sempre os mesmos. O número de execuções efetuadas é o mesmo adotado no Desafio ROADEF 2005.

Uma vez estabelecido o custo médio obtido por uma hiper-heurística para uma instância, é necessário traduzi-lo, procurando minimizar a perda de informações, para a forma de uma avaliação que esteja em intervalo independente dos custos das soluções e que possa ser diretamente comparada à avaliação de qualquer algoritmo, em qualquer instância. Para realizar essa tarefa, utilizamos a mesma técnica empregada no Desafio ROADEF 2005. A avaliação de uma hiper-heurística *hh* em relação a uma instância I será calculada como $10\,\frac{M_I - r_{hh_I}}{M_I - m_I}$, sendo r_{hh_I} o custo médio obtido por *hh* em I, M_I o maior dos custos médios obtidos em I nos experimentos e m_I o menor dos custos médios obtidos em I. Com isso, a avaliação de cada algoritmo, para cada instância, ficará no intervalo $[0,10]$ e, assim, a avaliação final de um algoritmo pode ser calculada como a média de suas avaliações em todas as instâncias (note que as avaliações maiores indicarão os melhores resultados).

Adotamos alguns referenciais externos para avaliar as hiper-heurísticas de forma menos restrita. Os custos advindos dessas fontes foram utilizados durante o cálculo das avaliações finais, de forma que, na prática, todos os limites inferiores m_I, definidos acima, foram obtidos a partir de origens externas. Abaixo, descrevemos as fontes utilizadas:

- No caso do problema do roteamento de veículos com restrições de capacidade, utilizamos o custo da melhor solução conhecida para cada uma das instâncias, segundo Cordeau et al. [11]. Na prática, quanto mais próximo um custo médio obtido neste trabalho estiver da melhor solução conhecida, maior será a avaliação da hiper-heurística correspondente.

- Para o problema da permutação de veículos, utilizamos, com finalidades distintas, dois referenciais:

  – Os resultados obtidos na fase final do Desafio ROADEF 2005 pela equipe constituída por Bertrand Estellon, Frédéric Gardi e Karim Nouioua. Essa equipe foi a

vencedora da competição e os custos obtidos por ela entrarão nos cálculos das avaliações, funcionando, na prática, como limitantes.

– Os resultados obtidos na fase final do Desafio ROADEF 2005 pela equipe constituída por Bin Hu e Gunnar W. Klau. Essa equipe ficou em último lugar na fase final da competição. O estágio inicial do Desafio ROADEF 2005 contou com 55 equipes, sendo 18 delas classificadas para a fase final. Os resultados da equipe de Hu e Klau, que não apenas entraram nos cálculos de avaliações que realizamos, como também receberam avaliações, funcionaram como indicadores da qualidade necessária para que um algoritmo esteja no nível dos principais participantes do Desafio ROADEF 2005.

Com o objetivo de simplificar nossas menções das hiper-heurísticas e dos referenciais que figuram, especialmente, nas tabelas de resultados que exibimos neste capítulo, adotamos a seguinte nomenclatura:

- *HHESv1*: nossa implementação (Seção 4.1.1) da hiper-heurística de Soubeiga.
- *HHESv2*: a primeira extensão a $HHESv1$. Trata-se da versão em que passamos a considerar o desempenho histórico de todas as seqüências compostas por 3 heurísticas de baixo nível. Esta alteração e as duas a seguir estão descritas na Seção 4.1.2.
- *HHESv3*: a segunda extensão que implementamos para a hiper-heurística de Soubeiga. Passamos, desta vez, a considerar também o desempenho histórico das seqüências compostas por 4 operadores.
- *HHESv4*: nossa última extensão à hiper-heurística de Soubeiga, que consiste em desconsiderar o tempo de execução das heurísticas de baixo nível durante as redefinições dos valores que indicam a qualidade desses operadores.
- *HHAGv1*: nossa implementação (Seção 4.2.1) do algoritmo genético hiper-heurístico de Cowling et al.
- *HHAGv2*: a primeira extensão que implementamos para $HHAGv1$. Nesta versão, introduzimos nosso operador de mutação baseado em otimização local direta.
- *HHAGv3*: a segunda extensão à hiper-heurística de Cowling et al., em que implementamos a primeira mudança descrita na Seção 4.2.3, alterando a função de aptidão.
- *HHAGv4*: a última versão implementada para a hiper-heurística de Cowling et al. Conforme descrito na Seção 4.2.3, deixamos de considerar o comprimento dos cromossomos ao avaliá-los.

- *HHED*: nosso algoritmo hiper-heurístico de estimação de distribuição, apresentado na Seção 4.3.
- *HHRC*: nossa hiper-heurística de reconexão de caminhos, apresentada na Seção 4.4.
- *HHDS*: nossa hiper-heurística mais simples, baseada em uma função de decisão, descrita na Seção 4.5.
- *Gardi*: o algoritmo utilizado pela equipe vencedora do Desafio ROADEF 2005.
- *Klau*: o algoritmo utilizado pela equipe que obteve a última colocação na etapa final do Desafio ROADEF 2005.

Como veremos na Figura 6.17 e na Tabela 6.4, não há razões para crer que as hiper-heurísticas *HHESv*2 e *HHESv*3 tenham constituído melhorias em relação a *HHESv*1. Por isso, a modificação implementada para dar origem a *HHESv*4 teve *HHESv*1 como ponto de partida. Por motivos análogos, a hiper-heurística *HHAGv*3 tomou *HHAGv*2 como base e a hiper-heurística *HHAGv*4 foi construída também a partir de *HHAGv*2.

Tendo como fim, mais uma vez, facilitar a leitura das tabelas de resultados, adotamos algumas siglas para nomear as instâncias utilizadas, conforme segue:

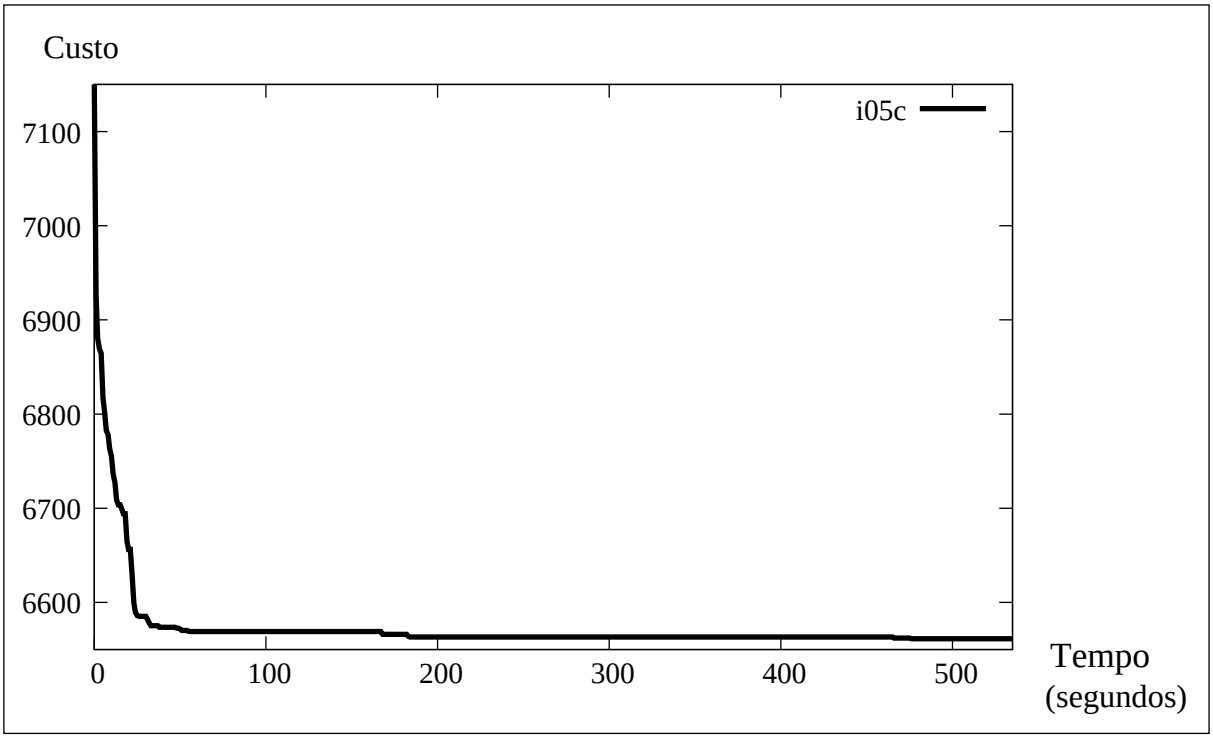
- *i01r* a *i16r* : referem-se às instâncias do conjunto *A* disponibilizado no Desafio ROADEF 2005. Na Tabela 6.1, relacionamos os nomes que adotamos às instâncias correspondendes, conforme a nomenclatura utilizada na competição.
- *i01c* a *i20c* : referem-se às instâncias de Golden et al., na ordem estabelecida pelos autores e utilizada, por exemplo, por Cordeau et al. [11].

Inicialmente, havíamos adotado, para os experimentos com o CVRP, o conjunto criado por Christofides e Eilon, com 15 instâncias disponíveis na Internet [63]. Porém, logo se tornou clara a conveniência de utilizarmos instâncias mais complexas, uma vez que os resultados obtidos pelos diferentes métodos aqui avaliados tendem a ser bastante similares e a ficarem próximos dos ótimos. Um menor número de soluções efetivamente alcançadas na prática teria como indesejável conseqüência a diminuição da variedade de avaliações para os algoritmos. De fato, mesmo com a adoção do conjunto de instâncias de Golden et al., pode-se notar que o resultado obtido por uma hiper-heurística após alguns segundos de execução é, em geral, bastante similar ao seu resultado final. A Figura 6.1 ilustra esse comportamento, mostrando o progresso da melhor solução conhecida pelo algoritmo, desde o início da execução da hiper-heurística (partindo

Tabela 6.1: Relação entre os Nomes das Instâncias.

Sigla neste trabalho	*Nome original no Desafio ROADEF 2005*
i01r	022_3_4_EP_RAF_ENP
i02r	022_3_4_RAF_EP_ENP
i03r	024_38_3_EP_ENP_RAF
i04r	024_38_3_EP_RAF_ENP
i05r	024_38_5_EP_ENP_RAF
i06r	024_38_5_EP_RAF_ENP
i07r	025_38_1_EP_ENP_RAF
i08r	025_38_1_EP_RAF_ENP
i09r	039_38_4_EP_RAF_ch1
i10r	039_38_4_RAF_EP_ch1
i11r	048_39_1_EP_ENP_RAF
i12r	048_39_1_EP_RAF_ENP
i13r	064_38_2_EP_RAF_ENP_ch1
i14r	064_38_2_EP_RAF_ENP_ch2
i15r	064_38_2_RAF_EP_ENP_ch1
i16r	064_38_2_RAF_EP_ENP_ch2

Figura 6.1: Evolução da Melhor Solução Durante a Execução (CVRP).



da solução obtida pelo método Savings). O gráfico foi traçado de forma a representar a média de diversas execuções (englobando todas as hiper-heurísticas) para a instância *i05c*.

As figuras 6.2 a 6.12 apresentam os resultados obtidos pelas hiper-heurísticas em suas execuções. Cada valor exibido representa o custo da solução fornecida pela hiper-heurística na execução considerada. Cada figura se refere a uma hiper-heurística e apresenta todos os resultados obtidos por ela tanto nas instâncias do conjunto *A* do Desafio ROADEF 2005 quanto nas instâncias de Golden et al. para o CVRP. A coluna intitulada como *Média* tem significado óbvio, enquanto a coluna C_v indica o coeficiente de variação (razão entre o desvio padrão e a média) observado para a hiper-heurística em cada instância, considerando-se os resultados das 5 execuções.

Naturalmente, o custo médio obtido por cada hiper-heurística em cada instância é o valor mais importante a ser observado nas figuras 6.2 a 6.12. No entanto, uma característica bastante desejável é que os coeficientes de variação sejam baixos, pois uma aplicação prática de um algoritmo de otimização consiste, em geral, em uma única execução para cada instância considerada e, portanto, o nível de qualidade do algoritmo deve ser previsível. Nesse sentido, as hiper-heurísticas obtiveram bons resultados no CVRP, com coeficientes de variação quase sempre abaixo de 2%. Esses coeficientes tendem a variar com as características das instâncias consideradas, especialmente no caso do problema da permutação de veículos, em que a função objetivo atribui pesos severamente diferentes aos diversos critérios considerados – conforme vimos na Seção 5.2.

Devido às discrepâncias entre os coeficientes de variação obtidos em instâncias distintas, a avaliação das hiper-heurísticas nesse aspecto foi efetuada de maneira comparativa. Para isso, adotamos estratégia semelhante à utilizada para a avaliação de qualidade, de forma que cada coeficiente foi transformado em uma avaliação no intervalo $[0,10]$, sendo o valor 10 uma indicação de que a hiper-heurística obteve o menor coeficiente de variação na instância e o valor 0 uma representação do maior coeficiente obtido por uma hiper-heurística na instância. A Tabela 6.2 mostra a média das avaliações de cada hiper-heurística nesse quesito, considerando os dois problemas separadamente. A coluna *Média* consiste na média aritmética entre as duas avaliações exibidas na tabela.

Em lugar do coeficiente de variação, pode-se utilizar o desvio padrão para o cálculo das variações de resultados que as hiper-heurísticas apresentam em cada instância. Com isso, ao transformarmos os desvios em avaliações no intervalo $[0,10]$, da mesma forma empregada anteriormente, as hiper-heurísticas de maior qualidade (de acordo com o custo médio das soluções) passam a receber avaliações ligeiramente superiores àquelas que obtiveram a partir do C_v, uma vez que este último é inversamente proporcional à média. A Tabela 6.3 exibe as avaliações baseadas no desvio padrão. Observe que os resultados são bastante similares aos anteriores, especialmente em relação à ordem em que as hiper-heurísticas aparecem na tabela.

Figura 6.2: Experimentos com a Primeira Versão da Hiper-Heurística de Soubeiga (HHESv1).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	61036	52069	57042	59048	60069	**57852,8**	*5,498%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	22135725	25147751	26147758	22128710	14149761	**21941941**	*19,202%*
i04r	22468229	23472320	26500276	17467247	15462200	**21074054,4**	*19,174%*
i05r	29100675	22097669	26093636	23091675	26085642	**25293859,4**	*9,816%*
i06r	20526667	20509718	17501657	25460693	20544675	**20908682**	*12,243%*
i07r	499765	499767	444775	457784	424767	**465371,6**	*6,444%*
i08r	266002	271267	273188	273071	272248	**271155,2**	*0,984%*
i09r	56379000	54379000	61358000	59374000	56397000	**57577400**	*4,295%*
i10r	68317000	68314000	68269000	68306000	68304000	**68302000**	*0,025%*
i11r	11216481	10188455	12217455	7154446	11166444	**10388656,2**	*16,747%*
i12r	10226281	9237203	12253231	12211238	8226197	**10430830**	*15,350%*
i13r	166869	162890	174880	170876	171884	**169479,8**	*2,463%*
i14r	41168	42128	41135	42150	41145	**41545,2**	*1,167%*
i15r	63440933	63441924	63439910	63441918	63439909	**63440918,8**	*0,001%*
i16r	27367078	27367112	27367095	27367078	27367074	**27367087,4**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	5846,410	5835,040	5942,420	5995,310	6041,480	**5932,13**	*1,366%*
i02c	8966,400	8947,970	8966,400	8966,400	8894,110	**8948,26**	*0,313%*
i03c	12087,200	12201,600	12223,000	12164,300	12225,900	**12180,4**	*0,423%*
i04c	15167,300	15763,400	15823,000	15860,200	15236,600	**15570,1**	*1,946%*
i05c	6552,990	6727,470	6676,200	6725,800	6500,870	**6636,67**	*1,401%*
i06c	9354,220	9082,130	9120,750	8979,480	9277,800	**9162,88**	*1,478%*
i07c	11097,600	11209,500	11316,300	11257,800	11241,000	**11224,44**	*0,644%*
i08c	12824,000	12819,000	12824,000	12812,400	12823,700	**12820,62**	*0,035%*
i09c	617,393	613,207	609,490	603,529	611,336	**610,99**	*0,746%*
i10c	782,742	789,256	786,757	792,466	781,011	**786,45**	*0,532%*
i11c	969,002	978,699	1001,950	988,796	1001,950	**988,08**	*1,310%*
i12c	1180,610	1204,780	1174,860	1179,440	1209,410	**1189,82**	*1,203%*
i13c	901,377	924,665	901,549	900,694	898,889	**905,43**	*1,067%*
i14c	1159,980	1159,980	1159,980	1145,970	1146,920	**1154,57**	*0,575%*
i15c	1426,660	1421,140	1416,590	1426,660	1420,460	**1422,3**	*0,273%*
i16c	1735,410	1734,820	1722,370	1720,860	1756,180	**1733,93**	*0,731%*
i17c	750,739	742,603	750,739	750,739	750,739	**749,11**	*0,434%*
i18c	1060,870	1060,870	1060,870	1060,870	1056,920	**1060,08**	*0,149%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1923,050	1923,050	1923,050	1923,050	1923,050	**1923,05**	*0,000%*

Figura 6.3: Experimentos com a Segunda Versão da Hiper-Heurística de Soubeiga (HHESv2).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	59040	57043	57051	59043	61049	**58645,2**	*2,552%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	27137772	21143786	21133761	18132729	24132706	**22336150,8**	*13,700%*
i04r	16474168	20495339	14477263	15460241	20455256	**17472453,4**	*14,491%*
i05r	33103695	25078656	22076647	25080669	31082639	**27284461,2**	*15,123%*
i06r	22520723	21499745	21497719	19526665	25514695	**22111909,4**	*8,858%*
i07r	456768	448775	472779	485776	505762	**473972**	*4,306%*
i08r	268998	273005	271192	266081	275055	**270866,2**	*1,151%*
i09r	60371000	64369000	55408000	59360000	49378000	**57777200**	*8,786%*
i10r	68302000	68320000	68306000	68298000	68301000	**68305400**	*0,011%*
i11r	14212469	10215450	9203442	12195460	12177451	**11600854,4**	*15,016%*
i12r	13235209	9228206	11240224	10235251	11264130	**11040604**	*12,043%*
i13r	166888	167864	169875	167881	167880	**168077,6**	*0,581%*
i14r	41161	44117	42166	42160	39129	**41746,6**	*3,886%*
i15r	63440897	63440909	63440922	63440932	63440925	**63440917**	*0,000%*
i16r	27367122	27367078	27367096	27367107	27367074	**27367095,4**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6064,730	6032,940	6007,140	5893,980	5860,840	**5971,93**	*1,339%*
i02c	8861,730	8946,520	8961,720	8966,400	8967,130	**8940,7**	*0,449%*
i03c	12277,200	12004,200	12020,400	12274,800	12017,100	**12118,74**	*1,060%*
i04c	15806,500	15437,000	15791,900	15445,200	15772,300	**15650,58**	*1,095%*
i05c	6650,530	6792,220	6776,040	6725,970	6838,060	**6756,56**	*0,947%*
i06c	9215,070	9030,070	9272,330	9018,480	9008,580	**9108,91**	*1,227%*
i07c	11455,800	11178,500	11400,500	11342,900	10956,200	**11266,78**	*1,606%*
i08c	12824,000	12821,400	12814,100	12824,000	12796,200	**12815,94**	*0,082%*
i09c	636,936	618,142	610,147	607,919	624,293	**619,49**	*1,694%*
i10c	775,736	799,105	792,316	779,273	789,874	**787,26**	*1,092%*
i11c	1001,950	971,656	1001,800	969,575	984,097	**985,82**	*1,422%*
i12c	1208,700	1202,270	1182,480	1213,270	1191,610	**1199,67**	*0,938%*
i13c	897,776	900,073	908,655	926,386	903,671	**907,31**	*1,126%*
i14c	1141,820	1153,730	1156,350	1147,530	1141,360	**1148,16**	*0,530%*
i15c	1421,630	1425,850	1416,320	1417,760	1413,940	**1419,1**	*0,296%*
i16c	1756,180	1753,010	1737,950	1746,330	1719,480	**1742,59**	*0,754%*
i17c	750,280	744,308	737,026	747,683	748,874	**745,63**	*0,635%*
i18c	1060,870	1060,870	1060,870	1060,870	1060,870	**1060,87**	*0,000%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1923,050	1923,050	1923,050	1923,050	1923,050	**1923,05**	*0,000%*

Figura 6.4: Experimentos com a Terceira Versão da Hiper-Heurística de Soubeiga (HHESv3).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	63043	57033	56052	59027	57070	**58445**	*4,267%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	21166775	23154801	25154747	25136723	27131730	**24348955,2**	*8,329%*
i04r	19456296	19469249	18471206	18456254	15458252	**18262251,4**	*8,058%*
i05r	25083682	37080635	24101696	21091610	29091625	**27289849,6**	*20,241%*
i06r	24502660	21425750	25528708	22496703	19532648	**22697293,8**	*9,439%*
i07r	439786	463790	489770	440790	459786	**458784,4**	*3,985%*
i08r	277043	276141	273970	269973	275016	**274428,6**	*0,895%*
i09r	59349000	57312000	63393000	54371000	61362000	**59157400**	*5,300%*
i10r	68298000	68301000	68291000	68298000	68294000	**68296400**	*0,005%*
i11r	11222469	11174448	11212455	9161459	12158434	**10985853**	*8,962%*
i12r	8263167	11264164	9232256	10250273	10260257	**9854023,4**	*10,377%*
i13r	160890	163881	168870	163861	172872	**166074,8**	*2,563%*
i14r	39138	41124	42115	38167	41139	**40336,6**	*3,605%*
i15r	63440906	63440901	63440914	63441908	63439922	**63440910,2**	*0,001%*
i16r	27367097	27367112	27367135	27367100	27367070	**27367102,8**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6004,650	6072,340	5847,540	5967,450	6040,280	**5986,45**	*1,300%*
i02c	8956,110	8966,400	8949,660	8965,790	8965,850	**8960,76**	*0,075%*
i03c	12266,500	12277,200	12277,200	12277,200	11945,100	**12208,64**	*1,080%*
i04c	15782,400	15860,200	15297,700	15868,100	15791,900	**15720,06**	*1,361%*
i05c	6757,730	6696,850	6664,020	6540,200	6828,260	**6697,41**	*1,441%*
i06c	9501,840	9006,550	9172,530	9020,430	9089,730	**9158,22**	*1,983%*
i07c	10925,900	11460,400	11348,000	11206,600	11461,000	**11280,38**	*1,776%*
i08c	12796,800	12817,700	12820,200	12694,200	12824,000	**12790,58**	*0,384%*
i09c	608,330	618,497	637,916	610,476	609,094	**616,86**	*1,805%*
i10c	798,020	792,762	799,663	789,226	795,345	**795**	*0,468%*
i11c	975,928	990,375	996,910	979,509	994,358	**987,42**	*0,837%*
i12c	1205,160	1210,790	1203,510	1201,180	1190,820	**1202,29**	*0,545%*
i13c	897,989	909,020	903,919	904,149	909,544	**904,92**	*0,463%*
i14c	1158,970	1144,450	1145,460	1152,160	1148,030	**1149,81**	*0,461%*
i15c	1415,470	1418,040	1425,680	1420,440	1413,130	**1418,55**	*0,305%*
i16c	1738,840	1749,120	1756,180	1727,790	1732,590	**1740,9**	*0,600%*
i17c	748,058	750,739	750,565	750,739	750,739	**750,17**	*0,141%*
i18c	1060,870	1060,870	1060,870	1060,870	1060,870	**1060,87**	*0,000%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1923,050	1923,050	1923,050	1923,050	1923,050	**1923,05**	*0,000%*

Figura 6.5: Experimentos com a Quarta Versão da Hiper-Heurística de Soubeiga (HHESv4).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	61042	58042	59079	59033	60068	**59452,8**	*1,717%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	25139742	25142734	23138738	32134722	24162786	**25943744,4**	*12,269%*
i04r	18454126	15439220	13473378	15459165	16451285	**15855434,8**	*10,214%*
i05r	25091669	23083655	28093705	24083647	21093655	**24289266,2**	*9,536%*
i06r	21521661	21521701	16508687	21496721	22533702	**20716494,4**	*10,333%*
i07r	393763	483768	432777	427793	459786	**439577,4**	*6,934%*
i08r	273100	269942	276140	275007	273036	**273445**	*0,772%*
i09r	55388000	63382000	67380000	71347000	60359000	**63571200**	*8,681%*
i10r	68300000	68299000	68326000	68268000	68273000	**68293200**	*0,031%*
i11r	11194458	8175448	11176453	10158442	8178445	**9776649,2**	*13,900%*
i12r	12240224	12242273	10246328	11234226	9271169	**11046844**	*10,465%*
i13r	162889	167893	158894	162874	165885	**163687**	*1,869%*
i14r	42155	43147	40176	38136	43122	**41347,2**	*4,680%*
i15r	63440912	63439911	63440910	63439927	63440939	**63440519,8**	*0,001%*
i16r	27367112	27367094	27367105	27367101	27367078	**27367098**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6009,320	6022,750	6037,290	6047,080	5885,710	**6000,43**	*0,980%*
i02c	8964,020	8946,290	8959,140	8948,050	8965,320	**8956,56**	*0,089%*
i03c	12227,000	12261,000	12133,400	12217,500	12144,200	**12196,62**	*0,406%*
i04c	15719,700	15258,400	15273,700	15018,600	15447,000	**15343,48**	*1,514%*
i05c	6512,270	6478,000	6849,840	6751,570	6723,200	**6662,98**	*2,157%*
i06c	9066,000	9343,440	9099,390	9085,570	9080,350	**9134,95**	*1,147%*
i07c	11143,400	11168,700	11164,200	11177,600	11413,500	**11213,48**	*0,897%*
i08c	12790,600	12824,000	12542,700	12824,000	12507,300	**12697,72**	*1,118%*
i09c	635,598	619,309	603,520	629,899	603,409	**618,35**	*2,139%*
i10c	789,473	781,812	773,710	793,343	799,663	**787,6**	*1,148%*
i11c	966,203	987,666	979,046	964,256	990,719	**977,58**	*1,105%*
i12c	1190,780	1209,890	1207,650	1212,280	1196,100	**1203,34**	*0,697%*
i13c	904,874	899,344	927,202	900,257	895,265	**905,39**	*1,251%*
i14c	1140,920	1156,130	1139,920	1159,980	1145,770	**1148,54**	*0,706%*
i15c	1425,850	1425,850	1419,600	1425,850	1425,850	**1424,6**	*0,175%*
i16c	1738,010	1743,650	1756,180	1754,610	1754,150	**1749,32**	*0,411%*
i17c	750,739	750,739	750,485	750,739	750,739	**750,69**	*0,014%*
i18c	1060,870	1060,870	1060,870	1060,870	1060,870	**1060,87**	*0,000%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1923,050	1923,050	1923,050	1923,050	1923,050	**1923,05**	*0,000%*

Figura 6.6: Experimentos com a Primeira Versão do Algoritmo Genético Hiper-Heurístico (HHAGv1).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	62047	67018	64013	65022	65013	**64622,6**	*2,500%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	26152771	33101664	27110690	26095644	31101688	**28712491,4**	*9,966%*
i04r	29401113	27358143	23367189	23343220	32362133	**27166359,6**	*12,865%*
i05r	32090704	34076549	32080582	33081597	29079572	**32081800,8**	*5,214%*
i06r	19441681	31409669	28427758	26393668	26412679	**26417091**	*14,915%*
i07r	496770	437745	429741	398748	401740	**432948,8**	*8,166%*
i08r	281029	277908	276809	280719	278206	**278934,2**	*0,593%*
i09r	78312000	81223000	80233000	71213000	81234000	**78443000**	*4,804%*
i10r	68279000	68304000	68343000	68321000	68294000	**68308200**	*0,032%*
i11r	13139453	14118418	11099394	14106419	15097395	**13512215,8**	*10,036%*
i12r	15195387	19190271	14206203	14196273	19200156	**16397658**	*14,105%*
i13r	179846	176835	184819	185825	177843	**181033,6**	*2,015%*
i14r	45105	46109	45074	46086	46125	**45699,8**	*1,091%*
i15r	63441872	63442883	63441891	63441872	63445881	**63442879,8**	*0,002%*
i16r	27367096	27367094	27367094	27367087	27367083	**27367090,8**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6074,500	6012,140	6008,210	6043,110	6032,180	**6034,03**	*0,397%*
i02c	8963,750	8969,330	8963,220	8946,900	8945,800	**8957,8**	*0,107%*
i03c	12259,500	12259,500	12277,200	12241,900	12277,200	**12263,06**	*0,108%*
i04c	15642,800	15680,400	15610,600	15829,900	15639,400	**15680,62**	*0,497%*
i05c	6751,570	6840,860	6849,840	6714,020	6466,680	**6724,59**	*2,067%*
i06c	9441,260	9416,720	9246,730	9022,810	9567,640	**9339,03**	*2,016%*
i07c	11370,700	11295,200	11343,600	11449,400	11210,700	**11333,92**	*0,700%*
i08c	12764,000	12822,500	12788,900	12710,200	12756,700	**12768,46**	*0,291%*
i09c	634,174	634,174	631,048	630,756	630,708	**632,17**	*0,259%*
i10c	799,663	795,031	799,663	798,048	795,099	**797,5**	*0,260%*
i11c	999,867	995,309	994,757	995,301	998,176	**996,68**	*0,200%*
i12c	1209,180	1209,180	1209,180	1207,180	1204,390	**1207,82**	*0,156%*
i13c	922,542	919,085	920,207	925,223	924,065	**922,22**	*0,249%*
i14c	1159,980	1159,980	1159,980	1151,390	1150,290	**1156,32**	*0,388%*
i15c	1425,020	1425,020	1425,020	1424,220	1425,020	**1424,86**	*0,022%*
i16c	1748,860	1742,600	1751,980	1745,620	1747,280	**1747,27**	*0,180%*
i17c	747,244	746,078	746,908	745,416	744,594	**746,05**	*0,130%*
i18c	1060,030	1060,030	1060,030	1060,030	1060,030	**1060,03**	*0,000%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1924,090	1922,390	1924,410	1922,390	1924,410	**1923,54**	*0,049%*

Figura 6.7: Experimentos com a Segunda Versão do Algoritmo Genético Hiper-Heurístico (HHAGv2).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	67010	63018	66014	65016	63016	**64814,8**	*2,465%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	28110653	40100673	26100706	31111686	31088658	**31302475,2**	*15,303%*
i04r	28361217	20362182	23362224	26356169	26372178	**24962794**	*11,217%*
i05r	27089600	26076603	32067599	29078583	26079591	**28078395,2**	*8,105%*
i06r	21419618	28412616	25413640	20404658	20403741	**23210854,6**	*13,743%*
i07r	446745	374746	392750	361742	403758	**395948,2**	*7,382%*
i08r	278695	281038	280773	279921	276896	**279464,6**	*0,545%*
i09r	71241000	80216000	88207000	81244000	84227000	**81027000**	*6,943%*
i10r	68302000	68291000	68308000	68304000	68311000	**68303200**	*0,010%*
i11r	12107410	10114400	12115419	14106381	13112418	**12311205,6**	*10,758%*
i12r	16199224	13195288	18197304	13193317	17194251	**15595876,8**	*13,210%*
i13r	180822	183817	176831	182827	180818	**181023**	*1,324%*
i14r	47079	48133	44092	45059	46115	**46095,6**	*3,100%*
i15r	63441872	63441878	63441874	63439894	63440873	**63441278,2**	*0,001%*
i16r	27367096	27367100	27367097	27367080	27367080	**27367090,6**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6032,720	6008,590	6019,070	6057,110	6012,870	**6026,07**	*0,291%*
i02c	8942,450	8935,390	8963,100	8935,390	8906,730	**8936,61**	*0,202%*
i03c	12259,500	12076,300	12277,200	12070,400	11914,900	**12119,66**	*1,111%*
i04c	15690,600	15607,300	15822,100	15793,100	15721,400	**15726,9**	*0,485%*
i05c	6693,160	6680,020	6708,440	6714,020	6668,680	**6692,86**	*0,254%*
i06c	9040,600	8942,320	8991,940	9455,710	9084,890	**9103,09**	*2,006%*
i07c	11280,900	10951,500	11392,900	11385,900	11201,100	**11242,46**	*1,440%*
i08c	12696,200	12753,100	12764,000	12764,000	12792,500	**12753,96**	*0,249%*
i09c	621,430	620,830	630,638	624,339	626,081	**624,66**	*0,568%*
i10c	799,663	799,663	799,663	792,164	799,663	**798,16**	*0,376%*
i11c	994,325	996,956	981,999	995,396	994,153	**992,57**	*0,542%*
i12c	1204,070	1208,250	1204,270	1206,030	1204,820	**1205,49**	*0,128%*
i13c	922,747	917,482	913,149	912,842	900,468	**913,34**	*0,807%*
i14c	1159,980	1145,120	1159,630	1151,160	1151,390	**1153,46**	*0,490%*
i15c	1419,160	1425,020	1425,020	1424,150	1424,700	**1423,61**	*0,158%*
i16c	1749,120	1734,480	1756,180	1746,270	1741,250	**1745,46**	*0,419%*
i17c	745,386	744,531	744,969	746,878	745,416	**745,44**	*0,106%*
i18c	1060,030	1060,030	1060,030	1060,030	1060,030	**1060,03**	*0,000%*
i19c	1438,380	1438,380	1438,060	1438,380	1438,380	**1438,32**	*0,009%*
i20c	1922,390	1924,410	1924,410	1924,410	1924,090	**1923,94**	*0,041%*

Figura 6.8: Experimentos com a Terceira Versão do Algoritmo Genético Hiper-Heurístico (HHAGv3).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	64025	63023	62020	62011	63016	**62819**	*1,197%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	26096687	26102651	23104685	30085661	28100664	**26698069,6**	*8,716%*
i04r	28357122	23347180	29367188	18361234	30347148	**25955974,4**	*17,330%*
i05r	27083585	31087602	24079578	35082556	31080607	**29682785,6**	*12,716%*
i06r	30436669	25392701	25417691	24420622	33405644	**27814665,4**	*12,585%*
i07r	384733	375752	388764	399739	406748	**391147,2**	*2,803%*
i08r	281599	279032	280807	282723	282346	**281301,4**	*0,466%*
i09r	81198000	74217000	82198000	83211000	85215000	**81207800**	*4,605%*
i10r	68325000	68311000	68311000	68313000	68300000	**68312000**	*0,012%*
i11r	15094365	15107393	13100398	13103388	12116396	**13704388**	*8,725%*
i12r	18191318	15203259	15200226	18195284	17194329	**16796883,2**	*8,052%*
i13r	178814	179814	182825	181822	178820	**180419**	*0,902%*
i14r	46082	47073	47064	46127	45096	**46288,4**	*1,590%*
i15r	63441868	63442872	63446872	63441887	63441883	**63443076,4**	*0,003%*
i16r	27367102	27367074	27367091	27367075	27367083	**27367085**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6056,510	6020,380	6052,670	6022,300	6039,370	**6038,25**	*0,247%*
i02c	8961,170	8938,190	8946,900	8964,360	8944,550	**8951,03**	*0,112%*
i03c	12277,200	12277,200	12277,200	12277,200	12259,500	**12273,66**	*0,058%*
i04c	15709,600	15817,900	15861,400	15584,900	15783,300	**15751,42**	*0,616%*
i05c	6712,020	6687,550	6723,000	6678,570	6703,050	**6700,84**	*0,240%*
i06c	9030,580	9292,510	9295,800	9208,240	9406,630	**9246,75**	*1,353%*
i07c	11355,900	11222,900	11459,200	11287,400	11294,500	**11323,98**	*0,703%*
i08c	12799,400	12769,000	12817,200	12762,000	12792,500	**12788,02**	*0,158%*
i09c	630,490	634,174	632,595	632,121	626,785	**631,23**	*0,398%*
i10c	799,663	799,416	799,663	798,313	799,559	**799,32**	*0,064%*
i11c	998,153	995,995	994,842	997,924	998,176	**997,02**	*0,136%*
i12c	1209,180	1206,940	1209,840	1209,180	1207,350	**1208,5**	*0,094%*
i13c	913,943	922,656	926,592	915,799	922,331	**920,26**	*0,510%*
i14c	1159,980	1159,980	1159,980	1159,980	1159,980	**1159,98**	*0,000%*
i15c	1422,900	1420,210	1424,000	1424,920	1425,020	**1423,41**	*0,125%*
i16c	1756,180	1756,180	1748,100	1756,180	1756,180	**1754,56**	*0,184%*
i17c	746,021	744,727	746,878	746,848	747,272	**746,35**	*0,122%*
i18c	1060,030	1060,030	1060,030	1060,030	1060,030	**1060,03**	*0,000%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1924,410	1924,410	1924,410	1922,700	1923,230	**1923,83**	*0,038%*

Figura 6.9: Experimentos com a Quarta Versão do Algoritmo Genético Hiper-Heurístico (HHAGv4).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	59049	64010	65018	63019	70019	**64223**	*5,505%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	32102686	25115705	32118687	32095624	33103649	**30907270,2**	*9,452%*
i04r	31364185	23363175	20362226	25348203	22358226	**24559203**	*15,324%*
i05r	27083620	25078588	29073626	27078594	30082624	**27679410,4**	*6,299%*
i06r	24421673	29419651	27417665	25411685	31401657	**27614466,2**	*9,259%*
i07r	439750	376742	393755	380766	432736	**404749,8**	*6,526%*
i08r	281790	280826	281766	281558	282496	**281687,2**	*0,190%*
i09r	79248000	87228000	80237000	79212000	82219000	**81628800**	*3,681%*
i10r	68288000	68320000	68284000	68287000	68320000	**68299800**	*0,024%*
i11r	18100410	12113410	15110390	13112394	12116398	**14110600,4**	*16,123%*
i12r	21189195	19200178	19198298	16209179	16196216	**18398613,2**	*10,515%*
i13r	180826	184820	181828	181828	183819	**182624,2**	*0,803%*
i14r	49100	47075	45108	44098	46108	**46297,8**	*3,709%*
i15r	63442857	63442864	63444872	63442875	63440892	**63442872**	*0,002%*
i16r	27367068	27367082	27367083	27367082	27367084	**27367079,8**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6043,110	6081,480	6021,020	6074,500	6045,690	**6053,16**	*0,365%*
i02c	8957,670	8927,140	8944,860	8939,870	8923,640	**8938,64**	*0,138%*
i03c	12241,900	12189,800	12265,100	12152,400	12263,800	**12222,6**	*0,364%*
i04c	15639,500	15737,100	15490,800	15738,000	15590,800	**15639,24**	*0,598%*
i05c	6703,050	6691,330	6460,980	6659,390	6687,450	**6640,44**	*1,368%*
i06c	9015,260	8985,160	8900,490	8888,280	9203,190	**8998,48**	*1,258%*
i07c	11155,600	11443,400	11333,300	11265,200	10997,100	**11238,92**	*1,360%*
i08c	12664,800	12761,000	12710,300	12752,900	12752,200	**12728,24**	*0,285%*
i09c	628,919	613,354	615,421	631,303	624,931	**622,79**	*1,153%*
i10c	798,205	799,663	795,667	794,453	798,313	**797,26**	*0,239%*
i11c	994,155	992,013	981,350	992,856	988,029	**989,68**	*0,469%*
i12c	1205,050	1207,840	1202,290	1208,370	1209,180	**1206,55**	*0,211%*
i13c	925,209	919,189	897,303	913,568	915,820	**914,22**	*1,020%*
i14c	1146,160	1144,150	1152,640	1146,880	1159,980	**1149,96**	*0,500%*
i15c	1425,020	1421,720	1423,810	1417,170	1424,000	**1422,34**	*0,197%*
i16c	1746,040	1734,950	1749,280	1750,860	1756,180	**1747,46**	*0,404%*
i17c	744,462	744,870	746,848	747,326	746,908	**746,08**	*0,158%*
i18c	1060,030	1060,030	1060,030	1060,030	1060,030	**1060,03**	*0,000%*
i19c	1438,380	1438,380	1438,380	1438,380	1438,380	**1438,38**	*0,000%*
i20c	1924,090	1922,920	1922,920	1924,410	1922,700	**1923,41**	*0,036%*

Figura 6.10: Experimentos com o Algoritmo Hiper-Heurístico de Estimação de Distribuição (HHED).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	63027	67017	66012	66025	65022	**65420,6**	*2,068%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	23112758	27137753	24117730	23132749	30110761	**25522350,2**	*10,680%*
i04r	26381213	26376212	23384179	21393246	19382222	**23383414,4**	*11,776%*
i05r	26092636	21084654	26090656	33089686	19089668	**25089460**	*19,367%*
i06r	17423692	22430695	23443695	23445661	28438734	**23036495,4**	*15,210%*
i07r	465783	516761	462752	463766	463771	**474566,6**	*4,450%*
i08r	280967	280916	281161	281952	275294	**280058**	*0,861%*
i09r	74274000	83257000	74265000	74275000	74293000	**76072800**	*4,722%*
i10r	68327000	68320000	68307000	68338000	68309000	**68320200**	*0,017%*
i11r	11129429	15129440	14146458	15127449	10129433	**13132441,8**	*15,982%*
i12r	12197288	13196317	13198330	16200337	16198391	**14198132,6**	*11,792%*
i13r	181850	176830	176856	180844	180833	**179442,6**	*1,201%*
i14r	45121	47123	44114	45124	48081	**45912,6**	*3,179%*
i15r	63440883	63441888	63441876	63439891	63440888	**63441085,2**	*0,001%*
i16r	27367096	27367092	27367087	27367085	27367074	**27367086,8**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6007,86	6007,86	5963,18	6007,86	6007,86	**5998,92**	*0,298%*
i02c	8905,04	8890,29	8860,84	8886,03	8860,13	**8880,47**	*0,197%*
i03c	12068,5	12023,5	12102	12156,4	12124,3	**12094,94**	*0,379%*
i04c	15523,4	15266,6	15435,4	15461,4	15389,7	**15415,3**	*0,558%*
i05c	6460,98	6460,98	6460,98	6654	6654	**6538,19**	*1,446%*
i06c	8878,31	8970,66	8980,16	8996,07	9008,62	**8966,76**	*0,514%*
i07c	10925,1	10816,7	10946,6	10908,7	10870,6	**10893,54**	*0,420%*
i08c	12678,2	12657,7	12665,4	12636	12674,5	**12662,36**	*0,118%*
i09c	612,72	612,01	606,69	610,16	612,54	**610,82**	*0,370%*
i10c	779,17	782,95	788,65	781,7	778,67	**782,23**	*0,457%*
i11c	979,08	982,74	974,62	982,07	983,01	**980,3**	*0,323%*
i12c	1195,88	1193,71	1183,42	1191,62	1195,7	**1192,07**	*0,385%*
i13c	889,28	887,81	889,28	892,26	894,63	**890,65**	*0,276%*
i14c	1122,98	1127,71	1123,76	1126,54	1131,32	**1126,46**	*0,265%*
i15c	1417,61	1419,42	1420,02	1418,97	1419,24	**1419,05**	*0,056%*
i16c	1733,03	1712,22	1734,74	1732,53	1728,72	**1728,25**	*0,477%*
i17c	741,22	737,07	735,15	736,07	739,52	**737,81**	*0,304%*
i18c	1055,13	1055,34	1055,66	1050,77	1052,72	**1053,92**	*0,179%*
i19c	1437,79	1437,55	1437,4	1435,8	1435,5	**1436,81**	*0,067%*
i20c	1924,41	1921,23	1922,23	1922,14	1922,46	**1922,49**	*0,054%*

Figura 6.11: Experimentos com a Hiper-Heurística de Reconexão de Caminhos (HHRC).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	62047	62038	63030	60038	64034	**62237,4**	*2,128%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	26152771	21140777	23128790	20137768	29144785	**23940978,2**	*13,845%*
i04r	29401113	20388222	19404237	20406241	17393208	**21398604,2**	*19,389%*
i05r	32090704	26087680	25086656	27082676	30085693	**28086681,8**	*9,289%*
i06r	19441681	27456649	17443656	22446700	21441737	**21646084,6**	*15,593%*
i07r	496770	493769	490748	471766	481763	**486963,2**	*1,870%*
i08r	281029	280228	281964	281321	276452	**280198,8**	*0,698%*
i09r	78312000	82274000	74277000	73276000	84273000	**78482400**	*5,487%*
i10r	68279000	68297000	68312000	68328000	68319000	**68307000**	*0,025%*
i11r	13139453	11175452	12175450	12152449	11131430	**11954846,8**	*6,233%*
i12r	15195387	13198318	12191300	12200338	11207280	**12798524,6**	*10,578%*
i13r	179846	176846	174851	172869	181833	**177249**	*1,834%*
i14r	45105	46119	45122	47065	42143	**45110,8**	*3,662%*
i15r	63441872	63442875	63442879	63441880	63442883	**63442477,8**	*0,001%*
i16r	27367096	27367081	27367106	27367106	27367116	**27367101**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	6007,86	5990,23	6007,86	6007,86	5963,18	**5995,4**	*0,292%*
i02c	8894	8876,41	8863,99	8922,18	8867,74	**8884,86**	*0,240%*
i03c	11960,4	11961,8	11832,8	11997,2	12004,4	**11951,32**	*0,518%*
i04c	15538,6	15458,3	15374,1	15554,2	15490,4	**15483,12**	*0,415%*
i05c	6518,62	6460,98	6460,98	6460,98	6460,98	**6472,51**	*0,356%*
i06c	8878,39	9023,85	8866,75	8745,44	8823,31	**8867,55**	*1,026%*
i07c	10756,1	10841,9	10883,5	10817,3	10900,2	**10839,8**	*0,472%*
i08c	12571,5	12571,2	12590,7	12606	12645,5	**12596,98**	*0,219%*
i09c	615,98	611,82	606,79	601,79	612,91	**609,86**	*0,820%*
i10c	777,05	795,56	785,19	793,68	782,21	**786,74**	*0,886%*
i11c	985,92	994,19	984,47	992,99	995,39	**990,59**	*0,454%*
i12c	1208,25	1209,18	1209,18	1202,92	1209,18	**1207,74**	*0,202%*
i13c	897,95	907	900,27	903,48	899,23	**901,58**	*0,362%*
i14c	1142,01	1152,8	1145,54	1142,46	1146,21	**1145,8**	*0,337%*
i15c	1425,02	1425,02	1422,15	1425,02	1423,9	**1424,22**	*0,079%*
i16c	1755,46	1737,88	1755,71	1756,18	1737,59	**1748,56**	*0,506%*
i17c	736,63	739,67	737,91	743,61	744,24	**740,41**	*0,410%*
i18c	1060,03	1048,15	1057,66	1060,03	1040,41	**1053,26**	*0,738%*
i19c	1438,38	1438,38	1438,38	1438,38	1438,38	**1438,38**	*0,000%*
i20c	1922,7	1923,23	1922,39	1922,39	1924,41	**1923,02**	*0,039%*

Figura 6.12: Experimentos com a Hiper-Heurística Simples (HHDS).

Resultados com o Problema do Desafio ROADEF 2005

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01r	63058	64034	68037	62058	61040	**63645,4**	*3,790%*
i02r	11048005	11048005	11048005	11048005	11048005	**11048005**	*0,000%*
i03r	24135729	24123717	21115664	22110733	24150728	**23127314,2**	*5,516%*
i04r	18442274	19423294	15408271	19418231	17410184	**18020450,8**	*8,341%*
i05r	32092652	32071630	28096679	27096647	21097627	**28091047**	*14,395%*
i06r	18478752	25472730	21522648	19452694	16484682	**20282301,2**	*15,091%*
i07r	378769	416760	427758	391762	415765	**406162,8**	*4,444%*
i08r	272996	272038	273003	271873	275109	**273003,8**	*0,422%*
i09r	55326000	56370000	62325000	51305000	58334000	**56732000**	*6,375%*
i10r	68267000	68275000	68248000	68291000	68256000	**68267400**	*0,022%*
i11r	10154452	10173447	9187455	13168447	10164473	**10569654,8**	*12,804%*
i12r	10236299	10242216	10227221	9230188	10225243	**10032233,4**	*3,998%*
i13r	167880	162895	171890	165897	172863	**168285**	*2,205%*
i14r	44138	41145	41193	45141	45167	**43356,8**	*4,208%*
i15r	63439925	63441921	63441926	63440908	63440907	**63441117,4**	*0,001%*
i16r	27367169	27367082	27367171	27367115	27367087	**27367124,8**	*0,000%*

Resultados com o CVRP

I./E.	*Exec. 1*	*Exec. 2*	*Exec. 3*	*Exec. 4*	*Exec. 5*	**Média**	C_v
i01c	5880,29	5796,1	5833,68	5821,35	5738,4	**5813,96**	*0,802%*
i02c	8878,14	8679,88	8650,23	8863,96	8787,93	**8772,03**	*1,061%*
i03c	12049,6	11979	12034,3	12042,1	12008,1	**12022,62**	*0,216%*
i04c	14980,6	15120,4	15066,4	15076,1	15254,7	**15099,64**	*0,595%*
i05c	6460,98	6567,91	6460,98	6466,68	6531,69	**6497,65**	*0,679%*
i06c	8725,72	8681,54	8886,96	8549,69	8734,9	**8715,76**	*1,241%*
i07c	10966,1	10834,4	10642,8	10772,6	10800,4	**10803,26**	*0,964%*
i08c	12656,9	12495	12586	12588,2	12638,6	**12592,94**	*0,447%*
i09c	621,08	629,61	628,23	628,95	625,47	**626,67**	*0,499%*
i10c	796,7	799,66	799,66	799,66	799,64	**799,07**	*0,148%*
i11c	1000,17	1000,17	1000,17	1003,34	998,5	**1000,47**	*0,157%*
i12c	1211,96	1209,41	1206,13	1211,52	1211,96	**1210,2**	*0,185%*
i13c	929,57	927,2	926,06	929,57	929,57	**928,39**	*0,160%*
i14c	1179,69	1159,98	1179,69	1179,69	1159,98	**1171,81**	*0,824%*
i15c	1425,02	1425,84	1445,23	1425,84	1425,02	**1429,39**	*0,555%*
i16c	1767,78	1767,78	1756,18	1756,18	1756,18	**1760,82**	*0,323%*
i17c	748,44	748,44	748,44	748,44	748,44	**748,44**	*0,000%*
i18c	1060,87	1075,03	1060,87	1060,87	1060,87	**1063,7**	*0,532%*
i19c	1438,38	1457,84	1438,38	1438,38	1438,38	**1442,27**	*0,540%*
i20c	1923,05	1923,05	1923,05	1955,93	1923,05	**1929,63**	*0,682%*

Tabela 6.2: Avaliações Comparativas dos Coeficientes de Variação por Instância.

Hiper-heurística	*ROADEF*	*CVRP*	***Média***
HHAGv3	6,653	9,055	**7,854**
HHAGv1	5,090	8,111	**6,601**
HHED	5,001	7,981	**6,491**
HHAGv4	5,644	6,999	**6,321**
HHRC	5,267	7,187	**6,227**
HHAGv2	5,357	6,969	**6,163**
HHDS	4,897	5,735	**5,316**
HHESv3	5,524	5,037	**5,281**
HHESv4	4,645	4,822	**4,734**
HHESv2	5,025	3,986	**4,506**
HHESv1	3,396	4,776	**4,086**

Tabela 6.3: Avaliações Comparativas dos Desvios Padrões por Instância.

Hiper-heurística	*ROADEF*	*CVRP*	***Média***
HHAGv3	6,025	9,044	**7,534**
HHED	4,889	8,013	**6,451**
HHRC	5,367	7,220	**6,294**
HHAGv1	4,469	8,098	**6,283**
HHAGv4	4,965	7,005	**5,985**
HHAGv2	4,868	6,981	**5,924**
HHDS	5,416	5,783	**5,599**
HHESv3	5,875	5,039	**5,457**
HHESv4	5,232	4,835	**5,034**
HHESv2	5,631	3,983	**4,807**
HHESv1	4,388	4,788	**4,588**

Além dos resultados obviamente sugeridos pelas tabelas 6.2 e 6.3, como o comportamento mais estável (no que diz respeito à qualidade) do algoritmo genético hiper-heurístico e dos algoritmos $HHED$ e $HHRC$, cabe observar que essas tabelas indicam uma maior previsibilidade das hiper-heurísticas evolutivas indiretas em comparação com as hiper-heurísticas de busca direta por entornos. O motivo desse comportamento não nos parece claro, embora uma causa a ser considerada possa ser o fato de que cada indivíduo é avaliado, nas hiper-heurísticas evolutivas indiretas, através da aplicação de sua seqüência de heurísticas à melhor solução conhecida pelo algoritmo, o que talvez resulte em uma menor exploração do espaço de busca por esse tipo de hiper-heurística (o Capítulo 7 sugere que essa hipótese seja estudada).

A Figura 6.13 mostra, para cada hiper-heurística, a média dos resultados de suas execuções em cada instância do problema do Desafio ROADEF 2005. Também estão presentes na figura as

médias obtidas, na etapa final da competição, pelas equipes cujos resultados adotamos como referência. Dessa forma, é fácil visualizar e comparar os custos médios em cada instância. A Figura 6.14 foi construída de forma análoga, mas refere-se ao CVRP. A linha intitulada como *MIN* apresenta, para cada instância, o custo da melhor solução conhecida.

As figuras 6.15 e 6.16 se referem, respectivamente, ao problema da permutação de veículos e ao CVRP. Elas apresentam as avaliações que associamos a cada algoritmo, através do mapeamento do custo médio obtido, em cada instância, para um valor no intervalo $[0, 10]$, conforme discutimos anteriormente. A Figura 6.17, obtida diretamente a partir das duas anteriores, mostra a avaliação média de cada algoritmo, tanto no problema proposto no Desafio ROADEF 2005 quanto no CVRP. Sem dúvida, a Figura 6.17 contém os resultados numéricos mais importantes a serem observados neste capítulo, uma vez que estabelece a avaliação final de cada algoritmo, considerando os diferentes referenciais adotados para cada problema.

Os resultados na Figura 6.17 não permitem a comparação entre os algoritmos considerando ambos os problemas, uma vez que, para isso, seria necessário estabelecer apenas uma avaliação final para cada hiper-heurística, a partir de todas as 36 instâncias utilizadas. Devido a esse fato, calculamos também de uma segunda forma as avaliações das hiper-heurísticas em cada instância. Trata-se do mesmo cálculo realizado anteriormente, mas desconsiderando as referências *Klau* e *Gardi*, no caso do problema da permutação de veículos, bem como o menor custo conhecido, no caso do CVRP. As médias finais obtidas dessa forma aparecem na Tabela 6.4. É importante ressaltar, no entanto, que a Tabela 6.4 deve ser vista apenas como um complemento aos resultados apresentados na Figura 6.17, uma vez que as informações presentes nesta última figura foram obtidas através de cálculos que consideram todos os elementos de que dispomos aqui.

Tabela 6.4: Avaliações Comparativas das Hiper-Heurísticas.

Hiper-heurística	*ROADEF*	*CVRP*	***Média***
HHESv1	7,862	4,933	**6,398**
HHESv4	8,055	4,498	**6,277**
HHED	3,707	8,115	**5,911**
HHESv2	7,284	4,458	**5,871**
HHDS	7,258	4,063	**5,661**
HHESv3	7,330	3,910	**5,620**
HHRC	3,974	6,793	**5,384**
HHAGv2	3,484	3,447	**3,466**
HHAGv4	2,231	3,695	**2,963**
HHAGv3	2,495	2,165	**2,330**
HHAGv1	2,087	2,292	**2,189**

Figura 6.13: Comparação entre as Hiper-Heurísticas e os Referenciais Externos – Problema do Desafio ROADEF 2005

Alg. / Ins.	*i01r*	*i02r*	*i03r*	*i04r*	*i05r*
HHESv1	57852,8	11048005,0	21941941,0	21074054,4	25293859,4
HHESv2	58645,2	11048005,0	22336150,8	17472453,4	27284461,2
HHESv3	58445,0	11048005,0	24348955,2	18262251,4	27289849,6
HHESv4	59452,8	11048005,0	25943744,4	15855434,8	24289266,2
HHAGv1	64622,6	11048005,0	28712491,4	27166359,6	32081800,8
HHAGv2	64814,8	11048005,0	31302475,2	24962794,0	28078395,2
HHAGv3	62819,0	11048005,0	26698069,6	25955974,4	29682785,6
HHAGv4	64223,0	11048005,0	30907270,2	24559203,0	27679410,4
HHED	65420,6	11048005,0	25522350,2	23383414,4	25089460,0
HHRC	62237,4	11048005,0	23940978,2	21398604,2	28086681,8
HHDS	63645,4	11048005,0	23127314,2	18020450,8	28091047,0
Klau	58005,0	11047008,0	71118491,0	63376290,0	82165438,0
Gardi	**31001,0**	**11039001,0**	**4000306,0**	**4249083,0**	**4034309,0**

Alg. / Ins.	*i06r*	*i07r*	*i08r*	*i09r*	*i10r*
HHESv1	20908682,0	465371,6	271155,2	57577400,0	68302000,0
HHESv2	22111909,4	473972,0	270866,2	57777200,0	68305400,0
HHESv3	22697293,8	458784,4	274428,6	59157400,0	68296400,0
HHESv4	20716494,4	439577,4	273445,0	63571200,0	68293200,0
HHAGv1	26417091,0	432948,8	278934,2	78443000,0	68308200,0
HHAGv2	23210854,6	395948,2	279464,6	81027000,0	68303200,0
HHAGv3	27814665,4	391147,2	281301,4	81207800,0	68312000,0
HHAGv4	27614466,2	404749,8	281687,2	81628800,0	68299800,0
HHED	23036495,4	474566,6	280058,0	76072800,0	68320200,0
HHRC	21646084,6	486963,2	280198,8	78482400,0	68307000,0
HHDS	20282301,2	406162,8	273003,8	56732000,0	68267400,0
Klau	75477143,0	1704363,0	282221,0	84248000,0	68344000,0
Gardi	**4280079,0**	**99720,0**	**231452,0**	**13131000,0**	**68161000,0**

Alg. / Ins.	*i11r*	*i12r*	*i13r*	*i14r*	*i15r*	*i16r*
HHESv1	10388656,2	10430830,0	169479,8	41545,2	63440918,8	27367087,4
HHESv2	11600854,4	11040604,0	168077,6	41746,6	63440917,0	27367095,4
HHESv3	10985853,0	9854023,4	166074,8	40336,6	63440910,2	27367102,8
HHESv4	9776649,2	11046844,0	163687,0	41347,2	63440519,8	27367098,0
HHAGv1	13512215,8	16397658,0	181033,6	45699,8	63442879,8	27367090,8
HHAGv2	12311205,6	15595876,8	181023,0	46095,6	63441278,2	27367090,6
HHAGv3	13704388,0	16796883,2	180419,0	46288,4	63443076,4	27367085,0
HHAGv4	14110600,4	18398613,2	182624,2	46297,8	63442872,0	27367079,8
HHED	13132441,8	14198132,6	179442,6	45912,6	63441085,2	27367086,8
HHRC	11954846,8	12798524,6	177249,0	45110,8	63442477,8	27367101,0
HHDS	10569654,8	10032233,4	168285,0	43356,8	63441117,4	27367124,8
Klau	27135374,0	27202853,0	184775,0	39058,0	63443831,0	27369063,0
Gardi	**61291,0**	**175615,0**	**112759,0**	**34051,0**	**63423782,0**	**27367052,0**

Figura 6.14: Comparação entre as Hiper-Heurísticas e os Menores Custos Conhecidos – CVRP

H. / Ins.	*i01c*	*i02c*	*i03c*	*i04c*	*i05c*	*i06c*
HHESv1	5932,13	8948,26	12180,40	15570,10	6636,67	9162,88
HHESv2	5971,93	8940,70	12118,74	15650,58	6756,56	9108,91
HHESv3	5986,45	8960,76	12208,64	15720,06	6697,41	9158,22
HHESv4	6000,43	8956,56	12196,62	15343,48	6662,98	9134,95
HHAGv1	6034,03	8957,80	12263,06	15680,62	6724,59	9339,03
HHAGv2	6026,07	8936,61	12119,66	15726,90	6692,86	9103,09
HHAGv3	6038,25	8951,03	12273,66	15751,42	6700,84	9246,75
HHAGv4	6053,16	8938,64	12222,60	15639,24	6640,44	8998,48
HHED	5998,92	8880,47	12094,94	15415,30	6538,19	8966,76
HHRC	5995,40	8884,86	11951,32	15483,12	6472,51	8867,55
HHDS	5813,96	8772,03	12022,62	15099,64	6497,65	8715,76
MIN	**5627,54**	**8447,92**	**11036,22**	**13624,52**	**6460,98**	**8412,88**

H. / Ins.	*i07c*	*i08c*	*i09c*	*i10c*	*i11c*	*i12c*	*i13c*
HHESv1	11224,44	12820,62	610,99	786,45	988,08	1189,82	905,43
HHESv2	11266,78	12815,94	619,49	787,26	985,82	1199,67	907,31
HHESv3	11280,38	12790,58	616,86	795,00	987,42	1202,29	904,92
HHESv4	11213,48	12697,72	618,35	787,60	977,58	1203,34	905,39
HHAGv1	11333,92	12768,46	632,17	797,50	996,68	1207,82	922,22
HHAGv2	11242,46	12753,96	624,66	798,16	992,57	1205,49	913,34
HHAGv3	11323,98	12788,02	631,23	799,32	997,02	1208,50	920,26
HHAGv4	11238,92	12728,24	622,79	797,26	989,68	1206,55	914,22
HHED	10893,54	12662,36	610,82	782,23	980,30	1192,07	890,65
HHRC	10839,80	12596,98	609,86	786,74	990,59	1207,74	901,58
HHDS	10803,26	12592,94	626,67	799,07	1000,47	1210,20	928,39
MIN	**10195,56**	**11663,55**	**583,39**	**742,03**	**918,45**	**1107,19**	**859,11**

H. / Ins.	*i14c*	*i15c*	*i16c*	*i17c*	*i18c*	*i19c*	*i20c*
HHESv1	1154,57	1422,30	1733,93	749,11	1060,08	1438,38	1923,05
HHESv2	1148,16	1419,10	1742,59	745,63	1060,87	1438,38	1923,05
HHESv3	1149,81	1418,55	1740,90	750,17	1060,87	1438,38	1923,05
HHESv4	1148,54	1424,60	1749,32	750,69	1060,87	1438,38	1923,05
HHAGv1	1156,32	1424,86	1747,27	746,05	1060,03	1438,38	1923,54
HHAGv2	1153,46	1423,61	1745,46	745,44	1060,03	1438,32	1923,94
HHAGv3	1159,98	1423,41	1754,56	746,35	1060,03	1438,38	1923,83
HHAGv4	1149,96	1422,34	1747,46	746,08	1060,03	1438,38	1923,41
HHED	1126,46	1419,05	1728,25	737,81	1053,92	1436,81	1922,49
HHRC	1145,80	1424,22	1748,56	740,41	1053,26	1438,38	1923,02
HHDS	1171,81	1429,39	1760,82	748,44	1063,70	1442,27	1929,63
MIN	**1081,31**	**1345,23**	**1622,69**	**707,79**	**998,73**	**1366,86**	**1821,15**

Figura 6.15: Avaliações das Hiper-Heurísticas por Instância – Problema do Desafio ROADEF 2005.

Algoritmo / Instância	*i01r*	*i02r*	*i03r*	*i04r*	*i05r*	*i06r*	*i07r*	*i08r*
HHESv1	2,20	0,00	7,33	7,15	7,28	7,66	7,72	2,18
HHESv2	1,97	0,00	7,27	7,76	7,02	7,50	7,67	2,24
HHESv3	2,03	0,00	6,97	7,63	7,02	7,41	7,76	1,53
HHESv4	1,73	0,00	6,73	8,04	7,41	7,69	7,88	1,73
HHAGv1	0,23	0,00	6,32	6,12	6,41	6,89	7,92	0,65
HHAGv2	0,18	0,00	5,93	6,50	6,92	7,34	8,15	0,54
HHAGv3	0,76	0,00	6,62	6,33	6,72	6,69	8,18	0,18
HHAGv4	0,35	0,00	5,99	6,57	6,97	6,72	8,10	0,11
HHED	0,00	0,00	6,79	6,76	7,31	7,37	7,66	0,43
HHRC	0,92	0,00	7,03	7,10	6,92	7,56	7,59	0,40
HHDS	0,52	0,00	7,15	7,67	6,92	7,75	8,09	1,82
Klau	2,15	1,11	0,00	0,00	0,00	0,00	0,00	0,00

Algoritmo / Instância	*i19r*	*i10r*	*i11r*	*i12r*	*i13r*	*i14r*	*i15r*	*i16r*
HHESv1	3,75	2,30	6,19	6,21	2,12	3,88	1,45	9,82
HHESv2	3,72	2,11	5,74	5,98	2,32	3,72	1,45	9,78
HHESv3	3,53	2,60	5,96	6,42	2,60	4,87	1,46	9,75
HHESv4	2,91	2,78	6,41	5,98	2,93	4,04	1,65	9,77
HHAGv1	0,82	1,96	5,03	4,00	0,52	0,49	0,47	9,81
HHAGv2	0,45	2,23	5,48	4,29	0,52	0,17	1,27	9,81
HHAGv3	0,43	1,75	4,96	3,85	0,60	0,01	0,38	9,84
HHAGv4	0,37	2,42	4,81	3,26	0,30	0,00	0,48	9,86
HHED	1,15	1,30	5,17	4,81	0,74	0,31	1,37	9,83
HHRC	0,81	2,02	5,61	5,33	1,05	0,97	0,67	9,76
HHDS	3,87	4,19	6,12	6,35	2,29	2,40	1,35	9,64
Klau	0,00	0,00	0,00	0,00	0,00	5,91	0,00	0,00

Antes de tecermos os últimos comentários sobre a qualidade das hiper-heurísticas avaliadas neste trabalho, ressaltemos que outra característica desejável em um algoritmo heurístico de otimização é que seu nível de qualidade não varie excessivamente à medida em que o aplicamos a diferentes instâncias do mesmo problema. Para avaliarmos essa característica, em cada um dos problemas adotados, calculamos, para cada hiper-heurística, o desvio padrão das avaliações de qualidade que ela obteve no problema. Mais especificamente, obtivemos, para cada hiper-heurística, o desvio padrão de 16 avaliações, no caso do Desafio ROADEF 2005, e o desvio padrão de 20 avaliações, no caso do CVRP. Esses desvios aparecem na Figura 6.18, nas colunas intituladas como σ.

Além das conclusões obviamente sugeridas pela Figura 6.18 – a maior previsibilidade do algo-

Figura 6.16: Avaliações das Hiper-Heurísticas por Instância – CVRP.

Hiper-heuríst. / Inst.	*i01c*	*i02c*	*i03c*	*i04c*	*i05c*	*i06c*	*i07c*	*i08c*	*i09c*	*i10c*
HHESv1	2,84	0,24	0,75	0,85	4,06	1,90	0,96	0,00	4,34	2,25
HHESv2	1,91	0,39	1,25	0,47	0,00	2,48	0,59	0,04	2,60	2,11
HHESv3	1,57	0,00	0,53	0,15	2,00	1,95	0,47	0,26	3,14	0,75
HHESv4	1,24	0,08	0,62	1,92	3,17	2,20	1,06	1,06	2,83	2,05
HHAGv1	0,45	0,06	0,09	0,33	1,08	0,00	0,00	0,45	0,00	0,32
HHAGv2	0,64	0,47	1,24	0,12	2,16	2,55	0,80	0,58	1,54	0,20
HHAGv3	0,35	0,19	0,00	0,00	1,89	1,00	0,09	0,28	0,19	0,00
HHAGv4	0,00	0,43	0,41	0,53	3,93	3,68	0,83	0,80	1,92	0,36
HHED	1,27	1,57	1,44	1,58	7,39	4,02	3,87	1,37	4,38	2,98
HHRC	1,36	1,48	2,60	1,26	9,61	5,09	4,34	1,93	4,57	2,20
HHDS	5,62	3,68	2,03	3,06	8,76	6,73	4,66	1,97	1,13	0,04

Hiper-heuríst. / Inst.	*i11c*	*i12c*	*i13c*	*i14c*	*i15c*	*i16c*	*i17c*	*i18c*	*i19c*	*i20c*
HHESv1	1,51	1,98	3,31	1,91	0,84	1,95	0,37	0,56	0,52	0,61
HHESv2	1,79	1,02	3,04	2,61	1,22	1,32	1,18	0,44	0,52	0,61
HHESv3	1,59	0,77	3,39	2,43	1,29	1,44	0,12	0,44	0,52	0,61
HHESv4	2,79	0,67	3,32	2,57	0,57	0,83	0,00	0,44	0,52	0,61
HHAGv1	0,46	0,23	0,89	1,71	0,54	0,98	1,08	0,57	0,52	0,56
HHAGv2	0,96	0,46	2,17	2,03	0,69	1,11	1,22	0,57	0,52	0,52
HHAGv3	0,42	0,16	1,17	1,31	0,71	0,45	1,01	0,57	0,52	0,53
HHAGv4	1,32	0,35	2,05	2,41	0,84	0,97	1,07	0,57	0,52	0,57
HHED	2,46	1,76	5,45	5,01	1,23	2,36	3,00	1,50	0,72	0,66
HHRC	1,20	0,24	3,87	2,87	0,61	0,89	2,40	1,61	0,52	0,61
HHDS	0,00	0,00	0,00	0,00	0,00	0,00	0,52	0,00	0,00	0,00

ritmo genético hiper-heurístico no tratamento do CVRP e um comportamento similar por parte da hiper-heurística de Soubeiga ao lidar com o problema da permutação de veículos –, também cabe notar que as hiper-heurísticas de busca direta por entornos parecem ter atacado de forma mais robusta o problema da permutação de veículos, especialmente se considerarmos também seus resultados na Figura 6.17. Um resultado mais importante a ser observado é o comportamento previsível da hiper-heurística de Soubeiga, quando comparada às demais, em ambos os problemas.

Na Seção 4.1, propusemos algumas alterações para serem experimentadas na hiper-heurística de Soubeiga, tendo as duas primeiras sido sugeridas pelo próprio autor em sua tese [57]. Com os resultados apresentados na Figura 6.17 e na Tabela 6.4, é possível avaliar o impacto dessas alterações, o que procuramos fazer abaixo:

Figura 6.17: Avaliações Finais das Hiper-Heurísticas.

Desafio ROADEF 2005

Algoritmo	*Avaliação*
HHESv4	4,855
HHESv3	4,846
HHESv1	4,828
HHESv2	4,765
HHDS	4,758
HHRC	3,983
HHED	3,813
HHAGv2	3,737
HHAGv1	3,602
HHAGv3	3,581
HHAGv4	3,518
Klau	0,573

CVRP

Hiper-heurística	*Avaliação*
HHED	2,701
HHRC	2,463
HHDS	1,910
HHESv1	1,587
HHESv4	1,427
HHESv2	1,279
HHAGv4	1,178
HHESv3	1,170
HHAGv2	1,027
HHAGv3	0,542
HHAGv1	0,516

Figura 6.18: Avaliações Comparativas dos Desvios Padrões por Problema.

Desafio ROADEF 2005

Hiper-Heurística	σ
HHESv3	2,801
HHESv2	2,817
HHESv1	2,820
HHESv4	2,862
HHDS	2,967
HHAGv1	3,205
HHRC	3,277
HHAGv2	3,301
HHAGv3	3,318
HHAGv4	3,333
HHED	3,334

CVRP

Hiper-Heurística	σ
HHAGv1	0,435
HHAGv3	0,492
HHAGv2	0,694
HHESv2	0,904
HHESv3	0,974
HHESv4	1,047
HHAGv4	1,063
HHESv1	1,231
HHED	1,755
HHRC	2,151
HHDS	2,596

- A Figura 6.17 sugere que a utilização de uma matriz de dimensão 3 para armazenar o desempenho histórico das seqüências de heurísticas não representa uma melhoria para a hiper-heurística de Soubeiga, uma vez que a avaliação final da hiper-heurística $HHESv2$ é inferior à de $HHESv1$ em ambos os problemas considerados.

- Parece-nos, ainda, que a utilização da matriz de dimensão 4 não é necessariamente uma melhoria em relação à hiper-heurística de Soubeiga, uma vez que, embora $HHESv3$ tenha

obtido resultados superiores a *HHESv*1 no problema da permutação de veículos, seus resultados foram inferiores no CVRP. Além disso, a Tabela 6.4 sugere um desempenho global superior para *HHESv*1.

- Resultado bastante similar pode ser observado para a versão *HHESv*4, ou seja, para o descarte do tempo de execução de cada operador nas avaliações efetuadas pela hiper-heurística de Soubeiga. No entanto, é importante notar que *HHESv*4 obteve a melhor avaliação, dentre todas as hiper-heurísticas, no tratamento do problema do Desafio ROADEF 2005 e que sua avaliação global está bastante próxima da obtida por *HHESv*1, de acordo com a Tabela 6.4.

No que diz respeito às alterações propostas, na Seção 4.2, para o algoritmo genético hiper-heurístico de Cowling et al., podemos observar que:

- As avaliações finais apontam uma superioridade de *HHAGv*2 em relação a *HHAGv*1, uma vez que *HHAGv*2 obteve melhores avaliações em ambos os problemas (Figura 6.17), aparecendo razoavelmente distante da versão original na Tabela 6.4. Com isso, podemos concluir que o novo operador de mutação, baseado em otimização local direta, representa uma melhoria para o algoritmo genético hiper-heurístico.

- Considerando-se que *HHAGv*3 foi implementada a partir de *HHAGv*2, não parece ter sido bem-sucedida a primeira alteração que experimentamos para a função de aptidão (dando tratamento diferente para os cromossomos com aptidão negativa), uma vez que os resultados obtidos por *HHAGv*3 foram claramente inferiores aos obtidos por *HHAGv*2.

- Resultado similar pode ser observado para *HHAGv*4, com base na Tabela 6.4. No entanto, cabe ressaltar que *HHAGv*4 se mostrou ligeiramente superior a *HHAGv*2 no CVRP, sugerindo que a função de aptidão que não leva em conta os comprimentos dos cromossomos não deve ser descartada em definitivo.

Na Seção 4.5, apresentamos uma hiper-heurística bastante simples, que chamamos de *HHDS*, baseada em uma função de decisão aleatória. Os resultados obtidos por essa hiper-heurística estão entre os mais importantes a serem observados neste trabalho, uma vez que o tempo requerido para implementá-la é muito inferior ao exigido pelas demais hiper-heurísticas – um dos benefícios trazidos por essa característica é a possibilidade de realização de experimentos futuros com facilidade. Na Figura 6.17, o algoritmo *HHDS* aparece entre as primeiras posições para ambos os problemas. A Tabela 6.4 reforça a idéia de que essa hiper-heurística teve desempenho bastante competitivo.

As avaliações obtidas pelas hiper-heurísticas de busca por entornos foram, em média, superiores às avaliações das demais hiper-heurísticas, sendo essa diferença mais clara no problema do Desafio ROADEF 2005 (conforme a Figura 6.17) e na Tabela 6.4. Mesmo numa comparação direta entre *HHDS* e *HHED*, não é adequado dizer que a segunda cumpriu com mais desenvoltura o papel que se espera de uma hiper-heurística: lidar de forma robusta com problemas de otimização distintos. Embora *HHED* apareça à frente de *HHDS* na Tabela 6.4, seu desempenho no problema da permutação de veículos foi insatisfatório, em comparação com as demais hiper-heurísticas. Parece razoável concluir que as hiper-heurísticas de busca direta por entornos se mostraram mais promissoras, especialmente se levarmos em conta que *HHDS* é um algoritmo bastante simples e, possivelmente, um representante menos valioso dessa classe de hiper-heurísticas. No entanto, os resultados obtidos por *HHED* e *HHRC* não podem ser desprezados: esses métodos alcançaram avaliações claramente superiores no CVRP.

A qualidade das hiper-heurísticas implementadas neste trabalho está evidenciada pelos resultados obtidos no problema do Desafio ROADEF 2005, uma vez que todas se mostraram consideravelmente superiores ao algoritmo desenvolvido pela equipe que obteve a última colocação na fase final da competição. Podemos afirmar que qualquer uma das hiper-heurísticas aqui apresentadas figuraria entre os finalistas do Desafio ROADEF 2005, se combinada ao conjunto razoavelmente simples de heurísticas de baixo nível que apresentamos na Seção 5.2.2. Trata-se de um resultado bastante interessante, uma vez que as hiper-heurísticas não são algoritmos específicos para o problema da permutação de veículos e que os participantes do Desafio ROADEF 2005 desenvolveram seus algoritmos durante cerca de 1 ano.

As soluções encontradas pelas hiper-heurísticas em suas execuções para lidar com o CVRP têm, em média, custo aproximadamente 7% mais alto que o da melhor solução conhecida para a respectiva instância. Considerando-se que essa distância se manteve abaixo de 17% em todas as execuções realizadas e que tal limite superior cai para 13% se excluirmos uma das instâncias (*i04c*), podemos concluir que as hiper-heurísticas obtiveram também resultados aceitáveis no tratamento desse problema de roteamento.

7 Trabalhos Futuros

A seguir, apontamos algumas direções que não tivemos a oportunidade de explorar plenamente neste trabalho e que, portanto, recomendamos como idéias a serem consideradas em trabalhos futuros que tenham as hiper-heurísticas como foco.

Uma possibilidade a ser analisada é a utilização de otimização local direta em alguns instantes da execução, visando a diminuir o custo da melhor solução conhecida pela hiper-heurística. Avaliamos essa técnica de maneira informal, empregando, em algumas execuções, um processo de otimização local a cada t segundos, sendo t um número positivo. Experimentos mais metódicos poderiam confirmar as observações a seguir. Notamos que o uso da otimização local, em geral, não melhora o resultado final do algoritmo. O aumento do tempo dedicado ao emprego dessa técnica (seja através da utilização de otimização local gulosa, da diminuição no valor de t ou do aumento no limite superior para o número de iterações que cada processo de otimização local realiza) teve apenas efeitos indesejáveis ou irrelevantes em relação aos resultados finais. Aparentemente, procurar diminuir o custo da solução de maneira imediatista produz um impacto global negativo. Note, ainda, que dificilmente a melhor solução conhecida pelo algoritmo em um momento avançado da execução poderá ser melhorada com um processo monótono.

Conforme apontamos na Seção 3.3.1, foram os experimentos efetuados com hiper-heurísticas triviais os responsáveis por motivar a implementação da principal hiper-heurística de Soubeiga de maneira que a aplicação de cada heurística de baixo nível selecionada pela função de decisão seja efetuada repetidamente, até que o operador não seja mais capaz de melhorar a solução corrente. Portanto, considerando-se que essa hiper-heurística apresentou-se superior a todas as demais em nossos experimentos, parece razoável que se avaliem, nas demais hiper-heurísticas, os resultados da adoção da mesma estratégia de Soubeiga para a aplicação de cada heurística de baixo nível.

Como todas as hiper-heurísticas evolutivas aqui apresentadas avaliam cada indivíduo através da aplicação de sua seqüência de heurísticas de baixo nível à melhor solução já encontrada, é possível que a capacidade de exploração dessas estratégias seja severamente limitada, pois, em determinadas situações, podem ser necessárias muitas modificações (considerando-se apenas

aquelas que os operadores utilizados são capazes de efetuar) para que uma solução de boa qualidade seja transformada em uma solução superior a ela. Eliminar essa limitação é um tema interessante a ser estudado, pois poderia representar um passo adiante das atuais hiper-heurísticas evolutivas indiretas, na comparação com as hiper-heurísticas de busca direta por entornos.

Outro experimento relevante talvez seja a combinação entre as duas principais classes de hiper-heurísticas que identificamos aqui. Poder-se-ia, por exemplo, construir uma hiper-heurística que realize uma busca direta por entornos em uma primeira fase de execução e, em seguida, utilize um algoritmo populacional indireto, iniciando-se com indivíduos construídos a partir de seqüências de heurísticas extraídas do processo de busca por entornos inicialmente executado (considerando-se que esse processo pode ser representado como uma seqüência de heurísticas).

Introduzimos neste trabalho um operador de mutação, baseado em otimização local direta, que demonstrou ter um bom impacto nos resultados de uma hiper-heurística evolutiva. Pode ser interessante, portanto, ampliar a utilização desses processos de otimização local no conjunto dos indivíduos. Uma tentativa natural, nesse sentido, seria utilizar tais processos para construir indivíduos completos (e não apenas para estender seqüências de heurísticas pré-existentes).

As estratégias utilizadas por meta-heurísticas de busca por entornos devem ser consideradas no desenvolvimento de hiper-heurísticas baseadas em funções de decisão. Com inspiração na busca tabu, por exemplo, podem-se adotar alguns critérios, como o impedimento de soluções repetidas (para isso, a hiper-heurística precisaria receber uma função que verifica se duas soluções são iguais, o que em geral pode ser implementado com facilidade), a verificação da freqüência com que cada heurística de baixo nível foi utilizada nas últimas iterações (informação que poderia servir como um critério de diversificação adicional), bem como quaisquer outras técnicas baseadas nessa bem-sucedida meta-heurística.

O conjunto de heurísticas de baixo nível utilizado por uma hiper-heurística deve gozar de grande diversidade – para que permita uma ampla exploração do espaço de busca – e conter heurísticas capazes de transformar soluções de maneiras que tendam a diminuir significativamente o custo. No entanto, é natural que haja muitos conjuntos de heurísticas com essas características desejáveis. É importante, portanto, que sejam realizados estudos empíricos utilizando diferentes conjuntos de heurísticas no tratamento de um mesmo problema, a fim de verificar o quanto cada hiper-heurística depende das heurísticas de baixo nível utilizadas. Com isso, analisar-se-á uma outra faceta da qualidade mais desejada nas hiper-heurísticas: a robustez.

Referências Bibliográficas

[1] B. BACKER, V. FURNON, P. KILBY, P. PROSSER, and P. SHAW. Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics. *Journal of Heuristics*, 6(4):501–523, 2000.

[2] R. BAI and G. KENDALL. An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-Heuristics. In *Proceedings of the Fifth Metaheuristics International Conference*, Kyoto, 2003.

[3] S. BALUJA and R. CARUANA. Removing the Genetics from the Standard Genetic Algorithm. In A. PRIEDITIS and S. RUSSELL, editors, *Proceedings of the International Conference on Machine Learning*, pages 38–46. Morgan Kaufmann, 1995.

[4] R. BATTITI. Reactive Search: Towards Self-Tuning Heuristics. In V. RAYWARD-SMITH et al., editors, *Modern Heuristic Search Methods*, pages 61–83. Wiley, 1996.

[5] R. BATTITI and G. TECCHIOLLI. The Reactive Tabu Search. *ORSA Journal on Computing*, 6:126–140, 1994.

[6] D. BEASLEY, D. BULL, and R. MARTIN. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993.

[7] J. BONET, C. ISBELL, and P. VIOLA. MIMIC: Finding Optima by Estimating Probability Densities. In *Advances in Neural Information Processing Systems*, volume 9, 1997.

[8] E. BURKE, G. KENDALL, R. O'BRIEN, D. REDRUP, and E. SOUBEIGA. An Ant Algorithm Hyper-Heuristic. In *Proceedings of the Fifth Metaheuristics International Conference*, Kyoto, 2003.

[9] E. BURKE, G. KENDALL, and E. SOUBEIGA. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[10] G. CLARKE and J. WRIGHT. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, 1964.

[11] J. CORDEAU, M. GENDREAU, A. HERTZ, G. LAPORTE, and J. SORMANY. New Heuristics for the Vehicle Routing Problem. In A. LANGEVIN and D. RIOPEL, editors, *Logistics Systems: Design and Optimization*, pages 279–297. Springer, New York, 2005.

[12] T. CORMEN, C. LEISERSON, R. RIVEST, and C. STEIN. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[13] P. COWLING, G. KENDALL, and L. HAN. An Investigation of a HyperHeuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In *Proceedings of the Congress on Evolutionary Computation (CEC2002)*, pages 1185–1190, Honolulu, May 2002.

[14] P. COWLING, G. KENDALL, and E. SOUBEIGA. A Hyperheuristic Approach to Scheduling a Sales Summit. In *Proceedings of the Third International Conference of Practice and Theory of Automated Timetabling (PATAT 2000). Lecture Notes in Computer Science*, volume 2079, pages 176–190. Springer-Verlag.

[15] P. COWLING, G. KENDALL, and E. SOUBEIGA. A Parameter-Free Hyperheuristic for Scheduling a Sales Summit. In *Proceedings of 4th Metaheuristics International Conference (MIC 2001)*, pages 127–131, Porto, July 2001.

[16] P. COWLING, G. KENDALL, and E. SOUBEIGA. Hyperheuristics: A Robust Optimisation Method Applied to Nurse Scheduling. In *7th PPSN Conference. Lecture Notes in Computer Science*, pages 851–860. Springer, 2002.

[17] P. COWLING, G. KENDALL, and E. SOUBEIGA. Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimization. In S. CAGONI et al., editors, *Applications of Evolutionary Computing: Proceedings of Evo Workshops 2002. Lecture Notes in Computer Science*, volume 2279, pages 1–10, Kinsale, April 2002. Springer-Verlag.

[18] M. DORIGO and T. STÜTZLE. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In F. GLOVER and G. KOCHENBERGER, editors, *Handbook of Metaheuristics*, chapter 9. Kluwer, 2003.

[19] G. DUECK and T. SCHEURER. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[20] H. FANG, P. ROSS, and D. CORNE. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. In A. COHN, editor, *Proceedings of ECAI 94: 11th European Conference on Artificial Intelligence*, pages 590–594. John Wiley and Sons, 1994.

[21] M. GAREY and D. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[22] F. GLOVER. A Template for Scatter Search and Path Relinking. In J. HAO et al., editors, *Lecture Notes in Computer Science*, volume 1363, pages 13–54. Springer, 1998.

[23] F. GLOVER and M. LAGUNA. *Tabu Search*. Kluwer, 1997.

[24] F. GLOVER, M. LAGUNA, and R. MARTÍ. Scatter Search. In A. GHOSH and S. TSUTSUI, editors, *Advances in Evolutionary Computation: Theory and Applications*, pages 519–537. Springer-Verlag, New York, 2003.

[25] F. GLOVER, M. LAGUNA, and R. MARTÍ. New Ideas and Applications of Scatter Search and Path Relinking. In G. ONWUBOLU and B. BABU, editors, *New Optimization Techniques in Engineering*. Springer, 2004.

[26] F. GLOVER, M. LAGUNA, and R. MARTÍ. Scatter Search and Path Relinking: Foundations and Advanced Designs. In G. ONWUBOLU and B. BABU, editors, *New Optimization Techniques in Engineering*. Springer, 2004.

[27] F. GLOVER and B. MELIÁN. Búsqueda Tabú. *Revista Iberoamericana de Inteligencia Artificial (Asociación Española de Inteligencia Artificial)*, 2(19):29–48, 2003.

[28] D. GOLDBERG. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[29] B. GOLDEN, E. WASIL, J. KELLY, and I. CHAO. Metaheuristics in Vehicle Routing. In T. CRAINIC and G. LAPORTE, editors, *Fleet Management and Logistics*, pages 33–56. Kluwer, Boston, 1998.

[30] J. GRATCH, S. CHEIN, and G. JONG. Learning Search Control Knowledge for Deep Space Network Scheduling. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 135–142, 1993.

[31] M. GUIGNARD. Lagrangian Relaxation. In P. PARDALOS and M. RESENDE, editors, *Handbook of Applied Optimization*, pages 465–474. Oxford University Press, 2002.

[32] L. HAN and G. KENDALL. An Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2230–2237, Canberra, December 2003.

[33] L. HAN and G. KENDALL. Guided Operators for a Hyper-Heuristic Genetic Algorithm. In *Australian Conference on Artificial Intelligence*, pages 807–820, 2003.

[34] L. HAN, G. KENDALL, and P. COWLING. An Adaptive Length Chromosome Hyper-heuristic Genetic Algorithm for a Trainer Scheduling Problem. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2002)*, pages 267–271, November 2002.

[35] P. HANSEN, N. MLADENOVIC, and J. PÉREZ. Búsqueda de Entorno Variable. *Revista Iberoamericana de Inteligencia Artificial (Asociación Española de Inteligencia Artificial)*, 2(19):77–92, 2003.

[36] E. HART, P. ROSS, and J. NELSON. Solving a Real-World Problem Using an Evolving Heuristically Driven Schedule Builder. *Evolutionary Computation*, 6(1):61–80, 1998.

[37] X. HU. Particle Swarm Optimization. http://www.swarmintelligence.org.

[38] H. KAUTZ and B. SELMAN. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In *Working Notes of the Workshop on Planning as Combinatorial Search*, pages 58–60, Pittsburg, 1998.

[39] S. KIRKPATRICK, C. GELATT Jr., and M. VECCHI. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, may 1983.

[40] P. KOSTUCH. The University Course Timetabling Problem with a 3-phase Approach. In *The 5th International Conference on the Practice and Theory of Automated Timetabling*, Pittsburgh, 2004.

[41] P. LARRAÑAGA, J. LOZANO, and H. MÜHLENBEIN. Algoritmos de Estimación de Distribuciones en Problemas de Optimización Combinatoria. *Revista Iberoamericana de Inteligencia Artificial (Asociación Española de Inteligencia Artificial)*, 2(19):149–168, 2003.

[42] A. LIM, J. LIN, and F. XIAO. Particle Swarm Optimization and Hill Climbing to Solve the Bandwidth Minimization Problem. In *The Fifth Metaheuristics International Conference (MIC2003)*, Kyoto, 2003.

[43] S. LIN. Computer Solutions of the Traveling Salesman Problem. *Bell Systems Technology Journal*, 44:2245–2269, 1965.

[44] H. LOURENÇO, O. MARTIN, and T. STÜTZLE. Iterated Local Search. In F. GLOVER and G. KOCHENBERGER, editors, *Handbook of Metaheuristics*, chapter 11. Kluwer, 2003.

[45] B. MELIÁN, J. PÉREZ, and J. VEGA. Metaheuristics: A Global View. *Revista Iberoamericana de Inteligencia Artificial (Asociación Española de Inteligencia Artificial)*, 2(19):7–28, 2003.

[46] S. MINTON. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer, 1988.

[47] N. MLADENOVIC. A Variable Neighborhood Algorithm – a New Metaheuristic for Combinatorial Optimization. In *Abstracts of Papers Presented at Optimization Days*, page 112, 1995.

[48] P. MOSCATO and C. COTTA-PORRAS. Una Introducción a los Algoritmos Meméticos. *Revista Iberoamericana de Inteligencia Artificial (Asociación Española de Inteligencia Artificial)*, 2(19):131–148, 2003.

[49] H. MÜHLENBEIN. The Equation for Response to Selection and its Use for Prediction. *Evolutionary Computation*, 5(3):303–346, 1998.

[50] C. PAPADIMITRIOU and K. STEIGLITZ. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Minneola, 1998.

[51] P. POMEROY. An Introduction to Particle Swarm Optimization, 2003. http://www.adaptiveview.com/articles/ipsop1.html.

[52] M. RESENDE and C. RIBEIRO. Greedy Randomized Adaptive Search Procedures. In F. GLOVER and G. KOCHENBERGER, editors, *Handbook of Metaheuristics*. Kluwer, 2003.

[53] P. ROSS, E. BURKE, E. HART, G. KENDALL, J. NEWALL, and S. SCHULENBURG. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In F. GLOVER and G. KOCHENBERGER, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.

[54] P. ROSS, S. SCHULENBURG, J. MARÍN-BLÁZQUEZ, and E. HART. Hyper-Heuristics: Learning to Combine Simple Heuristics in Bin-Packing Problems. In *Genetic and Evolutionary Computation Conference (GECCO 2002)*, New York, July 2002.

[55] S. RUSSELL and P. NORVIG. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[56] N. SAITOU and M. NEI. The Neighbor-Joining Method: A New Method for Reconstructing Phylogenetic Trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[57] E. SOUBEIGA. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, 2003.

[58] R. SUTTON and A. BARTO. *Reinforcement Learning : An Introduction*. MIT Press, Cambridge, 1998.

[59] H. TERASHIMA-MARÍN, P. ROSS, and M. VALENZUELA-RENDÓN. Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In W. BANZHAF et al., editors, *Proceedings of the GECCO-99 Genetic and Evolutionary Computation Conference*, pages 635–642. Morgan Kaufmann, 1999.

[60] C. VOUDOURIS and E. TSANG. Guided Local Search. In F. GLOVER and G. KOCHENBERGER, editors, *Handbook on Metaheuristics*, chapter 7. Kluwer, 2003.

[61] Metaheuristics Network. http://www.metaheuristics.net.

[62] Challenge ROADEF 2005. http://www.prism.uvsq.fr/∼vdc/ROADEF/CHALLENGES/2005.

[63] VRP Web. http://neo.lcc.uma.es/radi-aeb/WebVRP.