UNIVERSIDADE FEDERAL FLUMINENSE

MARCOS ANTONIO GUERINE RIBEIRO

Novas Heurísticas Baseadas em Simulated Annealing para Três Problemas de Otimização Combinatória

NITERÓI

2018

UNIVERSIDADE FEDERAL FLUMINENSE

MARCOS ANTONIO GUERINE RIBEIRO

Novas Heurísticas Baseadas em Simulated Annealing para Três Problemas de Otimização Combinatória

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Algoritmos e Otimização

Orientadora:
Isabel Cristina Mello Rosseti

Coorientador:
Alexandre Plastino de Carvalho

NITERÓI

2018

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

R484n Ribeiro, Marcos Antonio Guerine
Novas Heurísticas Baseadas em Simulated Annealing para Três Problemas de Otimização Combinatória / Marcos Antonio Guerine Ribeiro ; Isabel Cristina Mello Rosseti, orientadora ; Alexandre Plastino De Carvalho, coorientador. Niterói, 2018.
113 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.d.12438735740

1. Metaheurística. 2. Mineração de dados (Computação). 3. Otimização combinatória (Computação). 4. Produção intelectual. I. Rosseti, Isabel Cristina Mello, orientadora. II. De Carvalho, Alexandre Plastino, coorientador. III. Universidade Federal Fluminense. Escola de Engenharia. IV. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

Marcos Antonio Guerine Ribeiro

Novas Heurísticas Baseadas em *Simulated Annealing* para Três Problemas de Otimização Combinatória

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Algoritmos e Otimização

BANCA EXAMINADORA

Prof.ª Isabel Cristina Mello Rosseti – Orientadora, UFF

Prof. Alexandre Plastino de Carvalho – Coorientador, UFF

Prof.ª Simone de Lima Martins, UFF

Prof. Yuri Abitbol de Menezes Frota, UFF

Prof.ª Adriana Cesário de Faria Alvim, UNIRIO

Prof. Haroldo Gambini Santos, UFOP

Niterói
2018

Dedico este trabalho à minha esposa Jaqueline Bernardi Guerine.

Agradecimentos

Hoje escrevo esta seção com um leve sorriso no rosto, que me remete à época de minha graduação. Logo na primeira aula de Introdução à Ciência da Computação, o saudoso professor Giancarlo perguntou à nossa turma quem gostaria de obter o título de Doutor. Influenciado, talvez, pela defesa de Tese do meu irmão que acabara de acontecer, lembro, como se fosse hoje, que naquele dia eu levantei a mão. Embora não soubesse das dificuldades que encontraria a partir dali, sabia que era questão de tempo. Era um sonho. E eu o busquei a todo momento. Assim, agradeço inicialmente ao meu irmão Glaydston não só pela referência acadêmica, mas pelo incansável incentivo à minha formação. Não poderia deixar de mencionar os meus pais, que também não mediram esforços para que toda essa trajetória fosse possível, e minha esposa, com quem dividi todas as angústias e anseios.

Chegando à UFF, buscando orientação, fui apresentado à professora Isabel Rosseti. Que sorte a minha. Ótima pessoa, excelente orientadora. Cheguei perdido, saio completamente direcionado. Mérito da Isabel, que me ensinou que reuniões também podem ser descontraídas, e do professor Alexandre Plastino, que é certamente o melhor orientador. A Isabel acalmava, consolava, entendia, buscava alternativas. O Plastino discutia, dava sugestões, pensava, argumentava. E deu certo. Agradeço demais pelos ensinamentos, pela confiança, pela paciência. Orientação impecável. Deixo aqui uma menção honrosa ao Murilo, que se juntou ao nosso grupo como aluno em orientação e rapidamente se tornou não somente um parceiro de trabalho, mas um amigo. Fazer pesquisa com vocês foi realmente fácil. Muito obrigado!

Agradeço também ao suporte incondicional fornecido por toda a família *OptHouse*. Ed, Igor, Matheus, Pablo, Heder, Alan, Uéverton, Marcos Melo, João, Paulo, Jânio, Schneider, e todas as suas respectivas companheiras. A todos os amigos que fiz durante os anos de UFF, em especial à Renathinha, Eyder, Burlamaqui, Pedro González, Jean, Juliana, Diego, Julliany e ao Gleiph. Bons tempos, pessoal.

À secretaria da pós-graduação, pelo apoio. Aos membros da banca, pelos comentários e sugestões. À CAPES, pelo apoio financeiro. Acima de tudo, agradeço a Deus.

"Foi na rua do perdeu onde tudo começou,

programando e festejando *OptHouse* se formou.

Com os amigos reunidos na lapa vamos chapar,

voltaremos para casa quando o dia clarear.

Cantareira não me esqueço bons momentos ali passei,

e por fim termino a noite programando outra vez.

OptHouse! *OptHouse*!

Nem todos bebem. Mas o que bebem, bebem pra caralho!"

República OptHouse (2009-2019)

Resumo

Uma tendência bastante comum na área de otimização combinatória é a concepção de abordagens híbridas. A metaheurística *Data Mining* GRASP (DM-GRASP), que combina a metaheurística GRASP com técnicas de mineração de dados, foi proposta baseando-se na hipótese de que padrões extraídos de soluções subótimas representam características dessas boas soluções e, assim, poderiam ser usados para guiar a busca por melhores soluções. O DM-GRASP obteve sucesso em diversos problemas de otimização combinatória, tanto em termos de qualidade de solução, quanto em termos de tempo computacional. A hibridização de mineração de dados com metaheurísticas, no entanto, foi investigada somente em heurísticas multipartidas, principalmente as baseadas na metaheurística GRASP.

Neste contexto, o primeiro objetivo deste trabalho é ampliar o escopo da hibridização de metaheurísticas com técnicas de Mineração de Dados, utilizando a metaheurística *Simulated Annealing* (SA) como base. O principal desafio deste trabalho é, considerando essa nova estrutura de metaheurística, definir o tipo de padrão a ser extraído, em que momento a mineração será realizada e, ainda, como utilizar os padrões em benefício da heurística. Para realizar este estudo, dois problemas de otimização combinatória foram considerados: o Problema de Rotulação Cartográfica de Pontos e o Problema de Agrupamento Capacitado com Centro Geométrico. Nesses problemas, a proposta consiste em inserir o módulo de mineração de dados em duas heurísticas baseadas em SA já existentes na literatura e competitivas com o estado-da-arte dos respectivos problemas.

O segundo objetivo desta Tese é explorar um novo problema de otimização combinatória, denominado Problema de Dispersão de Arquivos em Serviços de Armazenamento em Nuvem. Esse problema teve origem em uma aplicação real, cujo objetivo é preservar a confidencialidade de dados gerados a partir da execução de *workflows*. Para isso, deve-se criar um plano de dispersão para distribuir os arquivos produzidos/consumidos por esses *workflows* em diversos serviços de armazenamento em nuvem, buscando minimizar os custos envolvidos no armazenamento e transferência de dados, respeitando a capacidade desses serviços e evitando que arquivos que contenham qualquer relação estrutural ou semântica entre si sejam alocados juntos. Assim, após a definição do problema no contexto de otimização combinatória, foram propostos um modelo matemático utilizando programação linear inteira e duas heurísticas baseadas nas metaheurísticas GRASP e SA.

Palavras-chave: Metaheurística, *Simulated Annealing*, GRASP, Mineração de Dados, Otimização Combinatória

Abstract

A common trend in the combinatorial optimization research area is the design of hybrid approaches. The Data Mining GRASP (DM-GRASP) metaheuristic, which combines a GRASP metaheuristic with data mining techniques, was proposed based on the hypothesis that patterns extracted from sub-optimal solutions represent characteristics of these good solutions and could be used to guide the search for better solutions. DM-GRASP metaheuristic achieved success on several combinatorial optimization problems, both in terms of solution quality and computational time. The hybridization of data mining with metaheuristics, however, was only investigated with multistart heuristics, specially those based on GRASP metaheuristic.

In this context, the first objective of this thesis is to expand the scope of the hybridization of metaheuristics with data mining techniques, using the simulated annealing (SA) metaheuristic as the basis. The main challenge of this work is to define, considering this new metaheuristic structure, the type of pattern to be extracted, at which moment the data mining process should be performed and, also, how to use the mined patterns to help the heuristic search. In order to do this, two combinatorial optimization problems were considered: the point-feature cartographic label placement problem and capacitated centred clustering problem. In these problems, we propose the insertion of the data mining module into two state-of-the-art existing SA-based heuristics.

The second objective of this work is to explore a new combinatorial optimization problem, named Preserving Results Confidentiality in Cloud-based Scientific Workflows Problem. This problem arised from a real application whose purpose is to preserve the confidentiality of data generated from the execution of workflows. For this, a dispersion plan must be created to distribute the files produced / consumed by these workflows in several cloud storage services, seeking to minimize the costs involved of storing and transferring data, respecting the capacity of these services and avoiding to place together files that contain any structural or semantic relationship between them. Thus, after defining the problem in the context of combinatorial optimization, we propose a mathematical model using integer linear programming and two heuristics based on both GRASP and SA metaheuristics.

Keywords: Metaheuristic, Simulated Annealing, GRASP, Data Mining, Combinatorial Optimization

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

GRASP	:	*Greedy Randomized Adaptative Search Procedures*;
DM-GRASP	:	Metaheurística GRASP Híbrida com Mineração de Dados;
VND	:	*Variable Neighborhood Descent*;
ILS	:	*Iterated Local Search*;
CS	:	*Clustering Search*;
SA	:	*Simulated Annealing*;
LRC	:	Lista Restrita de Candidatos (*Restricted Candidate List*);
MD	:	Mineração de Dados (*Data Mining*);
MCF	:	Mineração de Conjuntos Frequentes (*Frequent Itemset Mining*);
CE	:	Conjunto Elite;
PRCP	:	Problema de Rotulação Cartográfica de Pontos;
CCCP	:	*Capacitated Centred Clustering Problem*;
PDASAN	:	Problema de Dispersão de Arquivos em Serviços de Armazenamento em Nuvem;

Sumário

Capítulo 1

Introdução

Metaheurísticas formam um importante conjunto de técnicas que resolvem problemas de otimização combinatória, obtendo soluções aproximadas desses problemas em um tempo computacional aceitável [37]. Embora grande parte dos problemas de otimização pertençam à classe $\mathcal{NP}$-difícil – ou seja, algoritmos polinomiais que os resolvam de forma exata não são conhecidos –, a qualidade das soluções encontradas por heurísticas baseadas nessas metaheurísticas tendem a se aproximar da solução ótima, o que fez com que elas passassem a ser amplamente utilizadas e investigadas nas últimas décadas.

Cada metaheurística é projetada de acordo com um paradigma distinto e oferece diferentes mecanismos para escapar de ótimos locais. Uma tendência bastante comum nessa área de pesquisa é a combinação de uma ou mais componentes de diferentes metaheurísticas com o intuito de aproveitar as vantagens que cada uma tem a oferecer [14]. Conceitos e processos de outras áreas de pesquisa também têm sido utilizados em conjunto com metaheurísticas, tais como técnicas de Mineração de Dados [47].

De acordo com Han e Kamber [47], Mineração de Dados (MD) consiste basicamente na extração de conhecimento de maneira automática, na forma de regras e padrões, de bases de dados de um domínio específico. Em MD, alguns tipos de regras e padrões podem ser destacados, tais como as regras de associação, padrões sequenciais, agrupamento de dados e conjuntos frequentes. As técnicas que mineram esses padrões, geralmente aplicadas no contexto de análise de grandes bases de dados para auxiliar no processo de tomada de decisão, podem ser aproveitadas também no contexto da otimização combinatória. Diversos algoritmos, baseados tanto em metaheurísticas como em métodos exatos, exploram uma grande quantidade de soluções no espaço de busca antes de retornar a melhor solução (ótima ou subótima). Essas soluções, se armazenadas, podem constituir uma importante base de dados que contém informações relevantes as quais, se devidamente

extraídas, podem ser utilizadas ao longo da busca por novas e melhores soluções.

Assim, Ribeiro *et al.* [92] propuseram a incorporação de técnicas de mineração de conjuntos frequentes (MCF) à metaheurística *Greedy Randomized Adaptive Search Procedures* (GRASP) [30], baseando-se na hipótese de que padrões extraídos de soluções subótimas obtidas por heurísticas baseadas na metaheurística GRASP poderiam representar características dessas boas soluções e, portanto, poderiam ser usados para guiar a busca por melhores soluções. A combinação de técnicas de MCF com a metaheurística GRASP foi inicialmente aplicada ao problema de empacotamento de conjuntos (PEC), a fim de avaliar o impacto da utilização de padrões na busca realizada em uma heurística baseada em GRASP. Os resultados reportados no trabalho de Ribeiro *et al.* [92] revelaram uma melhoria tanto em termos de qualidade de solução, quanto em termos de tempo computacional. Esses resultados, além de promissores, foram muito importantes pois deram início a uma linha de pesquisa que já vem sendo explorada há mais de dez anos.

A metaheurística *Data Mining* GRASP (DM-GRASP), como ficou conhecida essa combinação da metaheurística GRASP com técnicas de MCF, foi novamente aplicada com sucesso a outros problemas de otimização combinatória, tais como: o problema da maximização da diversidade [99], o problema de replicação de servidores para transmissão *multicast* confiável [99], o problema das p-medianas [67, 83] e, mais recentemente, ao problema de projeto de redes a 2-caminhos [11].

Esses trabalhos citados possuem uma característica em comum: as soluções dos problemas abordados são representadas por conjuntos de elementos e, portanto, não levam em consideração a ordem em que esses elementos ocorrem na solução. Em Plastino *et al.* [83], no problema das p-medianas, por exemplo, a ordem das facilidades escolhidas em cada solução não influencia no cálculo do custo na função objetivo. Entretanto, em alguns problemas de otimização combinatória, essa ordem tem papel importante, como é o caso dos problemas de roteamento de veículos e de escalonamento de tarefas.

Assim, a linha de pesquisa que explora a hibridização de MD com técnicas heurísticas obteve um novo avanço com o trabalho de Guerine *et al.* [44]. Nesse trabalho, a metaheurística DM-GRASP foi explorada no problema do caixeiro viajante com coleta e entrega envolvendo um único tipo de produto, problema cuja ordem dos clientes é relevante não apenas para definir a qualidade, como também a viabilidade das soluções. Experimentos computacionais comprovaram novamente o benefício da hibridização da metaheurística com MD, encontrando soluções melhores em um menor tempo computacional que a heu-

rística original.

Entretanto, outra característica em comum dessas abordagens híbridas com mineração de dados ainda limitava o escopo de sua aplicação. Nos trabalhos propostos, as heurísticas eram, em grande parte, limitadas a um escopo similar ao da metaheurística GRASP, com estrutura multipartida e iterações independentes, divididas em etapas de construção e de busca local. No trabalho de Martins *et al.* [67], ainda que a heurística não fosse baseada em GRASP, a estrutura do seu algoritmo também seguia um formato multipartida, com iterações independentes, realizando uma construção seguida de uma etapa de busca local. Nesse trabalho, a hibridização também obteve resultados relevantes, principalmente ao reduzir o tempo computacional de uma heurística já muito eficiente e estado-da-arte para o problema. No trabalho de Barbalho *et al.* [11], a heurística era baseada em GRASP, mas dispunha também de um mecanismo de memória, implementado por meio da componente de reconexão por caminhos. Mesmo com essa componente de adição de memória ao GRASP, a incorporação de MD ainda conseguiu bons resultados se comparados à heurística original, conseguindo inclusive melhorar os resultados da literatura para o problema abordado, o que reforça o benefício da utilização dos padrões.

O trabalho de Maia *et al.* [64] também teve um papel importante na continuidade dessa linha de pesquisa que combina técnica de MD com metaheurísticas. Em seu trabalho, assim como em Guerine *et al.* [44], Maia *et al.* [64] exploraram um problema cuja caracterização da solução era uma sequência de elementos – problema de roteamento de veículos com frota heterogênea –, propondo a incorporação de MD em uma heurística multipartida baseada na metaheurística *Iterated Local Search* (ILS) para a resolução desse problema. Além da combinação ter obtido ótimos resultados, conseguindo reportar inclusive novas melhores soluções para o problema, Maia [63] também propôs uma redução das instâncias do problema com base nos padrões minerados, resultando em instâncias simplificadas. Experimentos comprovaram que essa estratégia inédita, dentre todas as versões híbridas com MD, além de ter obtido ótimos resultados em relação à qualidade de solução para diversas instâncias, também reduziu o tempo computacional consumido em relação às outras abordagens híbridas.

Neste contexto, o primeiro objetivo desta Tese de Doutorado é ampliar ainda mais o escopo da hibridização de técnicas de Mineração de Dados com metaheurísticas, explorando essa combinação utilizando uma metaheurística que possua uma estrutura diferente de todas que foram adotadas até então. A metaheurística *Simulated Annealing* (SA), que será detalhada no Capítulo 2, foi escolhida como foco deste estudo. Embora seja classi-

ficada como "metaheurística baseada em busca local" [37], assim como o GRASP, o SA não possui um formato multipartida com iterações independentes. A metaheurística SA funciona em uma trajetória de execução que explora a vizinhança de uma única solução, buscando melhorá-la até que se alcance um ótimo local. Os principais desafios, portanto, são: (i) identificar em que momento da execução da metaheurística SA as soluções elite serão coletadas para a criação da base de soluções a serem mineradas, (ii) qual a estrutura dos padrões a serem minerados, (iii) em que momento esses padrões serão utilizados, e (iv) como utilizar as informações dos padrões em benefício do SA, levando-se em consideração as particularidades não só da metaheurística SA, mas também dos problemas de otimização a serem abordados. Para realizar este estudo e avaliar o impacto da utilização desses padrões durante a busca por melhores soluções em heurísticas baseadas em SA, dois problemas de otimização combinatória foram considerados: o Problema de Rotulação Cartográfica de Pontos (PRCP) e o Problema de Agrupamento Capacitado com Centro Geométrico (CCCP).

Nesses problemas (PRCP e CCCP), a proposta consiste em inserir o módulo de mineração de dados em duas heurísticas baseadas em SA já existentes na literatura e competitivas com o estado-da-arte dos respectivos problemas. Foram utilizadas, respectivamente, as heurísticas de Rabello *et al.* [85] para o PRCP e a de Chaves *et al.* [20] para o CCCP, como bases para as propostas de hibridização com MD desta Tese. Diversos experimentos computacionais foram conduzidos com instâncias conhecidas da literatura, a fim de verificar o desempenho das heurísticas propostas e entender como os padrões minerados puderam beneficiar as heurísticas.

O segundo objetivo desta Tese é explorar um novo problema de otimização combinatória, denominado Problema de Dispersão de Arquivos em Serviços de Armazenamento em Nuvem (PDASAN). O PDASAN, definido e explorado nesta Tese, foi idealizado com base em uma aplicação real e surgiu a partir de uma demanda de um grupo de pesquisa que trabalha com *workflows* científicos, liderado pelo Professor Daniel de Oliveira (IC-UFF). Nesse problema, o objetivo é preservar a confidencialidade de dados gerados a partir da execução de *workflows*, criando um plano de dispersão para distribuir os arquivos produzidos/consumidos por esses *workflows* em diversos serviços de armazenamento em nuvem, buscando minimizar os custos envolvidos no armazenamento e transferência de dados, mas evitando que arquivos que contenham qualquer relação estrutural ou semântica entre si sejam alocados juntos. Assim, como se trata de um problema novo na literatura, nesta Tese, inicialmente, o PDASAN é devidamente formalizado no contexto de otimização combinatória, considerando as suas características reais, seus objetivos e suas

restrições. Em seguida, um modelo matemático utilizando programação linear inteira é proposto e validado para várias instâncias, que foram geradas a partir de situações reais do problema. Por fim, duas heurísticas baseadas nas metaheurísticas GRASP e SA também são propostas e comparadas. Vários experimentos computacionais foram executados a fim de avaliar tanto o modelo matemático (a partir de sua implementação usando um resolvedor conhecido da literatura), como também as heurísticas propostas.

O restante deste documento está organizado da seguinte maneira. O Capítulo 2 introduz os principais conceitos e propriedades das metaheurísticas abordadas neste trabalho, além de também descrever os conceitos e técnicas de mineração de dados que serão utilizados ao longo da Tese. No Capítulo 3, o Problema de Rotulação Cartográfica de Pontos é apresentado, assim como a proposta de hibridização com MD e os resultados alcançados com essa abordagem. O Capítulo 4 apresenta o Problema de Agrupamento Capacitado com Centro Geométrico e detalha como a hibridização com MD foi conduzida para esse problema e, ao final, é feita uma análise de todos os experimentos que foram realizados. No Capítulo 5, um novo problema de otimização, denominado o problema de dispersão de arquivos em serviços de armazenamento em nuvem é discutido e formalizado, e uma modelagem matemática é proposta para o problema. Além disso, duas heurísticas também são descritas para resolver o problema e, ao final, todos os experimentos computacionais realizados são discutidos. Finalmente, o Capítulo 6 resume as contribuições e apresenta as conclusões desta Tese, juntamente com indicações de trabalhos futuros.

Capítulo 2

Metaheurísticas e Mineração de Dados

Problemas de otimização combinatória podem possuir alta dificuldade de resolução exata quando a quantidade de combinações de soluções é extremamente grande. No processo de busca pela solução ótima, dependendo do tamanho da instância, examinar todas as soluções acaba se tornando inviável. Para tais problemas, algoritmos baseados em metaheurísticas são propostos para examinar o espaço de busca de modo a encontrar soluções próximas às ótimas (ou até mesmo as ótimas) em um tempo computacional viável.

Neste capítulo, serão apresentados os principais conceitos das metaheurísticas GRASP, *Simulated Annealing* (SA) e *Clustering Search* (CS), bem como a conceitos importantes da área de Mineração de Dados (MD), visando facilitar o entendimento do processo de hibridização dessas metaheurísticas com as técnicas de MD.

2.1 Metaheurística GRASP

Greedy Randomized Adaptive Search Procedure (GRASP) é uma metaheurística multipartida, baseada em busca local, proposta por Feo e Resende [30], e que já foi aplicada com sucesso a diversos problemas de otimização combinatória [31, 32]. O GRASP é composto por um processo iterativo, no qual cada iteração é dividida em duas fases: construção e busca local. A cada iteração, uma solução viável s é construída por um procedimento guloso e aleatório e avaliada por uma função $f(s)$. Essa função, também conhecida como função objetivo, fornece o custo da solução construída e deve ser definida de acordo com o problema abordado. Em seguida, a solução s é submetida a uma busca local no espaço vizinho de soluções, com o objetivo de melhorá-la até encontrar um ótimo local. A melhor solução geral é mantida ao longo das iterações e retornada como resultado final.

O pseudocódigo do GRASP [30] para um problema de minimização é apresentado no Algoritmo 1. O primeiro passo do algoritmo é inicializar as soluções s^* (melhor solução) e s (solução corrente) como vazias e o valor do custo de s^*, representado por $f(s^*)$, com valor infinito. Assim, o laço principal começa e uma solução s é construída (linha 4). Essa solução é submetida à heurística de busca local (linha 5) e, caso o custo da solução s seja melhor (no caso, menor) que o custo da melhor solução s^*, então s^* é atualizada com a solução corrente s (linha 7). Esses passos se repetem até que se completem $maxIter$ iterações.

Algoritmo 1: Pseudocódigo do GRASP original para um problema de minimização

```
1: GRASP(α, E, maxIter)
2: s* ← s ← ∅; f(s*) ← ∞
3: para iter = 1 até maxIter faça
4:    s ← ConstruçãoGulosaAleatória(α, E)
5:    s ← BuscaLocal(s)
6:    se f(s) < f(s*) então
7:       s* ← s
8:    fim se
9: fim para
10: retorne s*
```

A etapa de construção é também iterativa e baseada em um procedimento guloso e aleatório, que considera o conjunto de todos os elementos E a serem inseridos na solução que será construída. Inicialmente, todos os itens $i \in E$ que podem ser inseridos na solução são avaliados de acordo com uma função $c(i)$, que determina o quanto aquele elemento contribui para a solução s. Assim, são inseridos em uma lista denominada Lista Restrita de Candidatos (LRC) todos os elementos i que atendem ao seguinte critério:

$$c(i) \in \left[c^{min}, c^{min} + \alpha\left(c^{max} - c^{min}\right)\right], \tag{2.1}$$

com $\alpha \in [0, 1]$ e considerando c^{min} e c^{max}, respectivamente, os valores máximo e mínimo de contribuição dentre os elementos avaliados. Além dessa, uma outra maneira de compor a LRC é por cardinalidade, onde uma quantidade predefinida de elementos (i.e., os p melhores elementos de acordo com $c(i)$) são adicionados à lista restrita de candidatos.

Após a criação da LRC, um dos elementos é escolhido aleatoriamente e inserido na solução. Em seguida, repetem-se esses mesmos passos até que todos os elementos possíveis sejam inseridos. Na Equação 2.1, pode-se notar que o parâmetro α funciona como uma componente de aleatoriedade. Quando α se aproxima de zero, a escolha tende a ser mais gulosa, ao passo que quando se aproxima de um, a seleção tende a ficar mais aleatória.

De maneira análoga, quando a LRC é construída por cardinalidade, quanto maior o valor de p, mais aleatória será a escolha, enquanto que quanto menor o valor de p, mais gulosa será a opção escolhida.

O Algoritmo 2 representa o pseudocódigo da fase de construção do GRASP. A solução a ser construída s é inicializada como uma solução vazia (linha 2) e, após avaliar cada elemento i, $\forall\ i \in E$ (linha 3), constrói-se a LRC (linha 5) de acordo com a função $c(i)$ e o parâmetro α, presentes na Equação 2.1. Dentre os elementos da LRC, um é escolhido aleatoriamente (linha 6) e inserido na solução (linha 7). Esses passos (linhas 5, 6, 7 e 8) se repetem até que uma solução seja construída.

Algoritmo 2: Pseudocódigo da fase de construção

```
1: ConstruçãoGulosaAleatória(α, E)
2: s ← ∅
3: Avaliar c(i) ∀ i ∈ E
4: enquanto ¬ SoluçãoCompleta(s) faça
5:     LRC ← ConstroiLRC(α)
6:     i ← EscolhaAleatória(LRC)
7:     s ← s ∪ {i}
8:     Avaliar c(i) ∀ i ∈ E\s
9: fim enquanto
10: retorne s
```

Após a construção, a solução s é submetida a uma fase de intensificação, com o objetivo de melhorá-la. A fase de busca local é composta por uma estratégia de aprimoramento iterativa, que busca soluções de melhor qualidade em um conjunto de soluções próximas a s. Esse conjunto é denominado vizinhança de s, representado por $N(s)$, e pode ser obtido aplicando operações simples (*e.g.*, permutações de elementos) que alteram s. Assim, a busca local visa encontrar o melhor vizinho (ou, em alguns casos, o primeiro melhor) de s com respeito a uma vizinhança $N(s)$.

O Algoritmo 3 apresenta o pseudocódigo da fase de busca local para o GRASP. A linha 2 define o conjunto H, composto por todas as soluções y presentes na vizinhança $N(s)$ que possuem qualidade melhor que a solução s, de acordo com a função de avaliação $f(s)$. Assim, a cada iteração, seleciona-se um vizinho s' que seja melhor que s (linha 4) e um novo conjunto H é estruturado com base na solução s' (linha 5). A busca local termina quando não existirem vizinhos capazes de melhorar a qualidade da solução corrente s'. Nesse caso, afirma-se que s' é ótimo local com respeito à vizinhança $N(s)$. Para mais informações sobre a metaheurística GRASP, outras aplicações, alternativas de construção e variações na etapa de busca local, ver [86].

Algoritmo 3: Pseudocódigo da fase de busca local

1: **BuscaLocal**(s)
2: $H = \{y \in N(s) | f(y) < f(s)\}$
3: **enquanto** $|H| > 0$ **faça**
4: Selecionar $s' \in H$
5: $H = \{y \in N(s') | f(y) < f(s')\}$
6: **fim enquanto**
7: **retorne** s'

2.2 Simulated Annealing

Simulated annealing (SA) é uma metaheurística baseada em busca local proposta por Kirkpatrick *et al.* [58] que possui um mecanismo de aceitação de solução o qual permite a exploração de soluções piores, com a intenção de escapar de ótimos locais [37]. Essa metaheurística parte de uma solução inicial e segue em uma trajetória única de evolução, realizando movimentos na solução corrente a fim de melhorá-la. Assim, a cada iteração do SA, uma nova solução é gerada a partir da solução atual, e as duas (a solução atual e a nova) são comparadas. As novas soluções geradas que possuem custos melhores sempre são aceitas. Porém, soluções cujos custos são piores também podem ser aceitas (*uphill moves*), mas dependem de um critério probabilístico que considera tanto o valor do parâmetro denominado temperatura, quanto a diferença entre as soluções. Como mencionado anteriormente, essa aceitação de piora no custo da solução funciona como um mecanismo para escapar de ótimos locais. No início do algoritmo, com a temperatura ainda alta, a aceitação desses movimentos de piora ocorre com uma certa frequência. À medida que o valor de temperatura do algoritmo vai se aproximando de zero, a probabilidade de aceitação de soluções piores diminui, até que essa aceitação seja praticamente nula. Nesse caso, ao atingir a temperatura de congelamento, a melhor solução é retornada como resultado do SA.

O Algoritmo 4 descreve um pseudocódigo do *Simulated Annealing* para um problema de minimização. Inicialmente, uma solução do problema é construída de forma aleatória (linha 2) e o valor de temperatura T é inicializado com a temperatura inicial T_0 (linha 4). Ao longo de uma execução completa da metaheurística SA, também conhecida como resfriamento, o valor de T vai sendo reduzido de acordo com a taxa de resfriamento ω ao final do laço principal (linha 16). Assim, para cada valor de temperatura, a solução atual é submetida a movimentos que fazem parte de uma estrutura de vizinhança $N(s)$. Movimentos dessa vizinhança que geram melhoria na solução são sempre aceitos, mas movimentos de piora são aceitos de acordo com um teste de probabilidade, baseado no algoritmo de Me-

trópolis [58]. Tal critério depende da temperatura corrente T e da diferença de custo das soluções sendo comparadas $\Delta(s, s')$ e a aceitação é feita da seguinte maneira. Um número r no intervalo $(0, 1)$ é gerado aleatoriamente e, caso satisfaça à condição $r < e^{\frac{-(\Delta(s,s'))}{T}}$, a piora na solução é aceita. Caso contrário, a solução s' é descartada e a busca continua. Dessa maneira, movimentos de piora têm uma chance de aceitação maior no início do algoritmo (i.e., quando a temperatura está alta), com o objetivo de diversificar a busca, mas essa chance diminui consideravelmente ao final do algoritmo (i.e., quando a temperatura está baixa) convergindo para um ótimo local. Movimentos que não alteram a função objetivo (movimentos de custo zero ou movimentos laterais) também podem ser aceitos, dependendo do problema.

Algoritmo 4 Pseudocódigo do Simulated Annealing para um problema de minimização

```
1: SA ( T_0, T_c, ω, SA_max)
2: s ← SoluçãoInicialAleatória();
3: s* ← s;
4: T ← T_0;
5: enquanto T > T_c faça
6:    iter ← 0;
7:    enquanto iter < SA_max faça
8:       iter ← iter + 1;
9:       s' ← N(s);
10:      se f(s') < f(s) então
11:         s ← s';
12:      senão
13:         s ← s', com probabilidade e^{-(f(s)-f(s'))/T};
14:      fim se
15:   fim enquanto
16:   T ← ωT;
17:   s* ← min(s*, s);
18: fim enquanto
19: retorne s*;
```

2.3 Clustering Search

O *Clustering Search* (CS) é uma metaheurística elaborada por Oliveira e Lorena [79] que procura identificar e explorar regiões promissoras no espaço de busca do problema de otimização combinatória, dividindo-o em agrupamentos denominados *clusters*. Essa ideia foi posteriormente generalizada para uma metaheurística híbrida por Oliveira *et al.* [78], combinando a etapa de agrupamento com uma heurística construtiva auxiliar para gerar soluções. A heurística geradora de soluções pode ser baseada em qualquer metaheurística,

tais como GRASP, *Simulated Annealing* (SA), Busca Tabu ou outra. Ao longo do CS, as soluções geradas são armazenadas nesses *clusters* respeitando a similaridade entre elas. Cada nova solução gerada deve ser incluída no *cluster* mais relacionado de acordo com uma métrica de distância. Cada *cluster* armazena uma solução central que o representa (a melhor solução associada ao agrupamento), e vai sendo preenchido com soluções até que um limiar de volume seja atingido. Nesse momento, considera-se que esse *cluster* indica um espaço promissor de busca e, então, um procedimento de busca local é aplicado à sua solução central.

Na metaheurística CS, portanto, os *clusters* são caracterizados por triplas $(\varsigma_i, \tau_i, \beta_i)$, cujas componentes correspondem, respectivamente, ao centro do *cluster*, seu volume e um indicador de ineficiência. O centro do *cluster* ς_i é a solução que representa o *cluster*. O volume τ_i define a quantidade de soluções que estão associadas ao *cluster* e o indicador de ineficiência β_i representa quantas iterações de busca local foram aplicadas ao centro do *cluster* sem obter melhorias.

O pseudocódigo da metaheurística Clustering Search para um problema de minimização é apresentado no Algoritmo 5. De início, na linha 2, todos os *clusters* são inicializados com uma solução inicial. Em seguida, para cada iteração do laço principal, uma nova solução s é gerada (linha 4). Essa solução é submetida ao cálculo de distância para todas as soluções centrais de todos os *clusters* – linha 5 – e passa a ser associada ao cluster i cuja solução central ς_i possui a menor distância para s. Se a solução s for a melhor daquela região, ela passa a ser o centro do *cluster* (linha 6). A seguir, o volume τ_i do *cluster* i é incrementado (linha 7) e, caso tenha atingido o limiar τ_{max}, a busca local é ativada na solução central ς_i (linha 10). Caso a busca seja ativada por mais de β_{max} vezes, uma pertubação é aplicada à solução ς_i. Ao atingir o critério de parada do algoritmo, a melhor solução s^* é retornada como resultado.

Na seção seguinte, serão apresentados os conceitos e técnicas de Mineração de Dados utilizados nesta tese e também em trabalhos anteriores no processo de combinação de MD com metaheurísticas.

2.4 Mineração de Dados

O conceito de Mineração de Dados (MD) surgiu no final da década de 80 e consiste na extração de conhecimento de bases de dados, de maneira automática, na forma de padrões ou regras [47]. Esses padrões e regras, tais como conjuntos frequentes, padrões sequen-

Algoritmo 5 Pseudocódigo do *Clustering Search* para um problema de minimização

```
1: CS(γ, τ_max, β_max)
2: Gerar γ clusters iniciais por meio de uma heurística geradora de soluções;
3: enquanto critério de parada não satisfeito faça
4:    s ← gerarSoluçãoHeuristica();
5:    i ← arg min Distância(s,ς_i);
          i∈{1,...,γ}
6:    ς_i ← min(s, ς_i);
7:    τ_i ← τ_i + 1;
8:    se τ_i ≥ τ_max então
9:       τ_i = 0;
10:      ς_i ←Busca_Local(ς_i);
11:      se β_i ≥ β_max então
12:         ς_i ←Pertubação(ς_i);
13:         β_i = 0;
14:      fim se
15:   fim se
16:   s* ← min(s*, ς_i);
17: fim enquanto
```

ciais e regras de associação (RA), podem ser utilizados para entender comportamentos e auxiliar na tomada de decisões.

As regras de associação podem ser consideradas como uma das principais formas de conhecimento extraídas de bases de dados [47]. Minerar RAs é uma tarefa descritiva de MD, que tem como objetivo buscar associações entre itens que ocorrem juntos com frequência dentro da base de dados. Tais associações podem fornecer diferentes perspectivas na análise dos dados.

A tarefa de minerar RAs é geralmente dividida em duas etapas. A primeira, que demanda maior esforço computacional [41], é denominada mineração de conjuntos frequentes (MCF) e consiste em, a partir de um conjunto de transações, identificar todos os subconjuntos de itens que ocorrem com uma certa frequência nesse conjunto de transações. A segunda etapa fornece, para cada conjunto frequente extraído na primeira etapa, um conjunto de regras de associação.

Seja $\Phi = \{i_1, i_2, \ldots, i_n\}$ um conjunto de itens de um domínio de aplicação e D uma base de dados de transações, onde cada transação T é um subconjunto de itens de Φ. Seja A um subconjunto de itens de Φ. Uma transação T contém A se $A \subseteq T$. O suporte de A é o número de transações de D que contém A. Um conjunto A é denominado frequente se seu suporte é maior ou igual ao valor de suporte mínimo min_{sup}, i.e., se $sup(A) \geq min_{sup}$. O problema de MCF consiste em extrair todos os conjuntos frequentes

da base de dados D, dado um valor de suporte mínimo especificado como dado de entrada. Além disso, um conjunto de itens A é frequente maximal se A é frequente e não existe superconjunto U tal que $A \subset U$ e U seja frequente. Diversos algoritmos foram propostos para extrair eficientemente conjuntos frequentes e frequentes maximais, tais como Apriori, PatriciaMine e *FPmax** [6, 42, 48].

Para exemplificar alguns conceitos, a Tabela 2.1 apresenta uma base de dados exemplo com seis transações, envolvendo o conjunto de itens $\Phi = \{a, b, c, d, e\}$. É possível observar que o conjunto $\{a, b\}$ está presente em três transações (1, 2 e 4) das seis possíveis e, portanto, tem suporte igual a *três* (ou 50%). Já o conjunto $\{c, d, e\}$ tem suporte igual a *um* (ou 16.6%), pois ocorre apenas na transação 5.

ID	Transações
1	$\{a, b, c, d\}$
2	$\{a, b\}$
3	$\{b, c, e\}$
4	$\{a, b, c\}$
5	$\{b, c, d, e\}$
6	$\{c, d\}$

Tabela 2.1: Banco de dados de transações.

Além disso, considera-se conjunto frequente maximal aquele conjunto frequente que não é subconjunto de outro conjunto frequente. Assim, assumindo sup_{min} igual a três (ou 50%), o conjunto $\{a, b\}$ é frequente e também maximal, pois possui suporte três e não existe nenhum conjunto frequente que o contenha.

Em 1993, o problema de MCF foi introduzido por Agrawal *et al.* em [5], onde os autores propuseram o algoritmo AIS, que evita avaliar todos os $2^{|\Phi|}$ subconjuntos possíveis, considerando apenas os elementos que ocorrem em alguma transação. Assim, foi reduzido o esforço computacional dessa etapa, que ainda assim era muito grande.

No ano seguinte, o algoritmo *Apriori* [6] foi proposto e obteve grandes melhorias em relação a esse esforço computacional, utilizando a seguinte propriedade: todo conjunto de itens que contém um subconjunto não frequente, também não é frequente. Dessa maneira, diminuiu-se consideravelmente a quantidade de subconjuntos a serem avaliados.

Desde então, muitos algoritmos propostos para a MCF foram baseados no *Apriori*, aprimorando suas características negativas. Tais algoritmos, segundo Han *et al.* [48], ainda tem como principal ponto negativo a geração de um grande número de conjuntos candidatos a serem avaliados, especialmente quando o suporte mínimo é baixo.

No entanto, existem alguns algoritmos que extraem os conjuntos frequentes sem a geração de conjuntos candidatos e, consequentemente, em muitos casos, evitam o ponto negativo do *Apriori*, tais como o *PatriciaMine* [9, 41], *FP-growth* [48] e *FP-growth** [41, 42, 43], cuja variação conhecida como *FPmax** [41, 42, 43] fornece conjuntos de itens frequentes maximais. O algoritmo *FPmax** tem sido a principal técnica de MCF utilizada nos trabalhos que envolvem o DM-GRASP, a versão híbrida do GRASP com mineração de dados [11, 44, 64, 67, 83, 93, 99].

A seção a seguir apresenta os dois *frameworks* que combinam GRASP com técnicas de mineração de conjuntos frequentes.

2.5 GRASP com Mineração de Dados

O GRASP, em sua forma original [30], tem como característica não possuir qualquer tipo de memória a longo prazo, com exceção da melhor solução que é mantida ao longo das iterações. Isto se deve ao fato de o GRASP ser iterativo e multipartida, com iterações totalmente independentes que, a cada partida, inicia uma nova e aleatória execução do processo de busca [88]. Ideias de manter informações de iterações anteriores têm sido investigadas, como técnicas baseadas em memória adaptativa [33, 60, 62, 94] e construção de vocabulário [12, 39].

Dentre essas estratégias, Laguna e Martí [59] adaptaram a técnica de reconexão por caminhos, proposta por Glover [38, 40], para utilizá-la em conjunto com o GRASP. Essa técnica de intensificação pode ser entendida como uma memória a longo prazo quando inserida ao GRASP, pois mantém um conjunto elite de soluções durante toda a execução do algoritmo. A reconexão por caminhos explora todo o espaço intermediário entre duas soluções s_i e s_f, escolhidas desse conjunto. Em s_i, são introduzidos iterativamente atributos de s_f de maneira a transformar s_i em s_f. As soluções intermediárias são avaliadas e, eventualmente, contribuem com a busca por melhores soluções.

Além dessas estratégias, Santos *et al.* [98] exploraram a hibridização de um algoritmo evolucionário (AE) com uma técnica de mineração de dados, baseada no algoritmo *Apriori*, como método de solução para o problema de roteamento de veículos com rota única (em inglês, *single-vehicle routing problem*). Em sua proposta, os autores extraíram padrões (a partir de um conjunto elite de soluções) que foram usados para construir e inserir novos indivíduos na população mantida pelo AE.

Essas abordagens, caracterizadas pela utilização de conjuntos de boas soluções, refor-

çam a ideia de que padrões extraídos desses conjuntos podem auxiliar na busca por melhores soluções. As próximas seções apresentam os conceitos das estratégias DM-GRASP e MDM-GRASP, duas abordagens que combinam a metaheurística GRASP com a técnica de mineração de conjuntos frequentes.

2.5.1 DM-GRASP

Proposto por Ribeiro *et al.* [92, 93], a estratégia *Data Mining GRASP* (DM-GRASP) baseia-se na hipótese de que padrões minerados a partir de um conjunto de soluções subótimas podem ser utilizados em procedimentos de construção adaptados, para guiar a exploração do espaço de busca, visando alcançar melhores soluções.

O DM-GRASP é dividido em duas etapas. A primeira, denominada fase de geração do conjunto elite (CE), consiste em executar k iterações do GRASP original e, a cada iteração, manter atualizado um conjunto com as d melhores soluções encontradas. Após essa etapa, o CE é submetido a um procedimento de mineração baseado na técnica de mineração de conjuntos frequentes, detalhada anteriormente. O CE funciona como uma base de dados de soluções e, assim, a mineração retorna os $numP$ maiores padrões que ocorreram com uma frequência mínima prefixada. Em outras palavras, cada padrão extraído é um conjunto de partes (ou elementos) frequentes dessas soluções.

Após a mineração, a segunda etapa do DM-GRASP, denominada etapa híbrida, consiste em executar mais k iterações do GRASP, mas agora substituindo a heurística construtiva original por uma heurística construtiva híbrida, adaptada para utilizar os padrões minerados na primeira etapa. A maneira clássica de seleção desses padrões na construção híbrida é escolher, de maneira sequencial, um padrão da lista de padrões minerados para ser aproveitado em cada iteração. Tão logo todos os padrões da lista forem utilizados, a escolha recomeça do primeiro padrão, similar à ideia de escalonamento de processos descrita pelo algoritmo de *round-robin*.

O Algoritmo 6 ilustra o pseudocódigo geral do DM-GRASP para um problema de minimização. No pseudocódigo, que é formado por dois laços com k iterações, o primeiro laço representa a etapa de geração do conjunto elite do DM-GRASP e o segundo laço representa a fase híbrida, em que a construção tradicional é substituída por uma construção adaptada. No primeiro, além da construção e da busca local (linhas 5 e 6, respectivamente), a construção do conjunto elite é feita na linha 10, atualizando o CE sempre que o conjunto não estiver totalmente preenchido ou quando a solução corrente for melhor que a pior das suas soluções. Entre os laços, acontece o processo de mineração (linha 12) e os

padrões minerados são armazenados na *ListaPadrões*.

Algoritmo 6: Pseudocódigo do DM-GRASP

```
1: DM-GRASP (α, E, k, sup_min, d, numP)
2: f(s*) ← f(s) ← ∞
3: CE ← ∅
4: para iter = 1 até k faça
5:    s ← ConstruçãoGulosaAleatória(α, E)
6:    s ← BuscaLocal(s)
7:    se f(s) < f(s*) então
8:       s* ← s
9:    fim se
10:   AtualizaCE(s, CE, d)
11: fim para
12: ListaPadrões ← ExecutaMineração(CE, sup_min, numP)
13: para iter = 1 até k faça
14:   p ← EscolhePadrão(ListaPadrões)
15:   s ← ConstruçãoAdaptada(α, E, p)
16:   s ← BuscaLocal(s)
17:   se f(s) < f(s*) então
18:      s* ← s
19:   fim se
20: fim para
21: retorne s*
```

No segundo laço, um padrão p é selecionado da *ListaPadrões* e aproveitado pela construção adaptada (linha 15), que substitui a construção original nesta fase híbrida do DM-GRASP. A busca local permanece idêntica à primeira etapa (linha 16).

O Algoritmo 7 descreve a construção adaptada para usar os padrões minerados. Um padrão p, dentre os padrões minerados para guiar a construção, é escolhido e passado como parâmetro para ser utilizado na construção. A solução inicial s recebe p como solução parcial (linha 2) e em seguida, a construção prossegue da mesma forma que a construção original, detalhada na Seção 2.1.

O processo de utilização do padrão (linha 2 do Algoritmo 7) depende do problema que está sendo tratado e de como sua solução é representada. Nos capítulos seguintes, será descrito em detalhes como os padrões foram de fato utilizados em cada um dos problemas abordados nesta Tese.

A próxima seção apresenta outra abordagem que combina MD com GRASP, porém aplicando a mineração de padrões mais de uma vez durante a execução do algoritmo.

Algoritmo 7: Pseudocódigo da construção adaptada

```
1: ConstruçãoAdaptada(α, E, p)
2: s ← p
3: Avaliar c(i) ∀ i ∈ E\s
4: enquanto ¬ SoluçãoCompleta(s) faça
5:    LRC ← ConstroiLRC(α)
6:    i ← EscolhaAleatoria(LRC)
7:    s ← s ∪ {i}
8:    Avaliar c(i) ∀ i ∈ E\s
9: fim enquanto
10: retorne s
```

2.5.2 MDM-GRASP

Na proposta híbrida DM-GRASP, o processo de mineração é executado somente uma vez, entre a primeira e a segunda etapa, exatamente na metade das iterações do algoritmo. Apesar do *framework* DM-GRASP [92, 99] ter alcançado resultados promissores, Plastino *et al.* [83] acreditaram que aplicar a mineração mais de uma vez poderia ser ainda mais eficiente. Essa ideia foi baseada na hipótese de que, se após a primeira extração e utilização dos padrões as soluções melhorassem, o conjunto elite seria formado por soluções cada vez mais refinadas e, por conseguinte, padrões cada vez mais refinados poderiam ser extraídos.

Assim, nessa nova abordagem, os autores definiram que a mineração seria aplicada tão logo o conjunto elite se tornasse estável, *i.e.*, o conjunto com as melhores soluções não fosse modificado num intervalo predefinido de iterações. Dessa maneira, a aplicação da mineração não está mais vinculada à quantidade de iterações (*e.g.*, metade das iterações do GRASP), mas sim condicionada à estabilidade do CE, podendo ser aplicada até mesmo antes da metade das iterações. Essa estratégia foi denominada pelos autores como *Multi Data Mining* GRASP (ou simplesmente MDM-GRASP).

O Algoritmo 8 mostra os passos do MDM-GRASP para um problema de minimização. No primeiro laço (linhas 4-12), são realizadas as etapas de construção e busca local (linhas 5 e 6, respectivamente), e o CE é preenchido com as d melhores soluções (linha 10). Esse laço somente é finalizado quando não há alterações no CE por um determinado número de iterações. Em seguida, inicia-se outro laço (linhas 13-25), em que a mineração é executada (linha 15) sempre que o CE se torna estável. Dessa maneira, na primeira iteração desse segundo laço, a mineração é sempre executada devido ao critério de parada do primeiro laço (por estabilidade do CE). Após a execução da mineração, seleciona-se um padrão (linha 17) que é usado na construção adaptada (linha 18) e, em seguida, aplica-se a busca

local (linha 19). Por fim, o CE é atualizado sempre que a solução corrente for melhor que a pior solução presente no CE.

Algoritmo 8: Pseudocódigo do MDM-GRASP

```
1: MDM-GRASP (α, E, maxIter, sup_min, d, numP)
2: f(s*) ← f(s) ← ∞; iter ← 1
3: CE ← ∅
4: enquanto CE ¬ estável faça
5:    s ← ConstruçãoGulosaAleatória(α, E)
6:    s ← BuscaLocal(s)
7:    se f(s) < f(s*) então
8:       s* ← s
9:    fim se
10:   AtualizaCE(s,CE,d)
11:   iter ← iter +1
12: fim enquanto
13: enquanto iter ≤ maxIter faça
14:   se CE estável então
15:      ListaPadrões ← ExecutaMineração(CE,sup_min, numP)
16:   fim se
17:   p ← EscolhePadrão(ListaPadrões)
18:   s ← ConstruçãoAdaptada(α,E,p)
19:   s ← BuscaLocal(s)
20:   se f(s) < f(s*) então
21:      s* ← s
22:   fim se
23:   AtualizaCE(s,CE,d)
24:   iter ← iter +1
25: fim enquanto
26: retorne s*
```

A estratégia MDM-GRASP foi inicialmente avaliada no problema das p-medianas [83], conseguindo superar os resultados do DM-GRASP (que, por sua vez, já havia superado o GRASP) em termos de qualidade de solução e também em tempo de execução. Em seguida, a estratégia foi avaliada no problema 2-PNDP [11], novamente conseguindo superar as versões GRASP e DM-GRASP. Nos dois casos, a versão MDM foi superior tanto em termos de qualidade de solução, quanto em termos de tempo computacional. Além disso, para algumas instâncias, a heurística MDM-GRASP ainda obteve os melhores resultados da literatura para o 2-PNDP. Diversos outros trabalhos também exploraram a estratégia *Multi-Data Mining*, tais como [44, 64, 67, 82] sempre com resultados superiores à estratégia DM-GRASP.

O Capítulo 3, a seguir, apresenta o problema de rotulação cartográfica de pontos (PRCP), que será utilizado como base para avaliar o comportamento da estratégia pro-

posta neste trabalho. Em seguida, alguns conceitos acerca desse problema são detalhados e uma formulação matemática do problema é descrita. Na sequência, detalha-se a heurística de Rabello *et al.* [85] para resolver o PRCP, que combina componentes das metaheurísticas *Simulated Annealing* e *Clustering Search*. Finalmente, a heurística híbrida com MD é apresentada, e todos os experimentos computacionais realizados comparando essas duas heurísticas são discutidos.

Capítulo 3

Problema de Rotulação Cartográfica de Pontos

3.1 Introdução

Posicionar rótulos em um mapa de tal forma que eles claramente representem os objetos ao qual estão associados é uma importante tarefa em cartografia. De acordo com Marks e Shieber [66], antes da automatização, a tarefa de rotulação despendia mais de 50% do tempo de produção de um mapa.

O Problema de Rotulação Cartográfica de Pontos (PRCP) é um problema de otimização combinatória amplamente estudado que consiste em atribuir rótulos de texto para cada objeto de um mapa, respeitando preferências e convenções cartográficas, tendo como objetivo evitar sobreposições entre diferentes rótulos. A ideia principal é prover clareza na visualização e entendimento do mapa a ser rotulado [85].

O PRCP é uma variação do *cartographic label placement problem* [51, 52], o qual normalmente abrange três diferentes tarefas: atribuir rótulos a objetos em formato de regiões (*e.g.*, continentes ou países), a objetos em formato de linha (*e.g.*, rios ou rodovias) e, por fim, a objetos em formato de ponto (*e.g.*, cidades ou hospitais). Essa última tarefa consiste no estudo de caso deste trabalho. Existem diversas aplicações do PRCP em cartografia automatizada, geoprocessamento e sistemas de informações geográficas [109].

Atingir um bom nível de visualização em um mapa está diretamente relacionado com a maneira de associar esses rótulos aos pontos do mapa. Cada ponto deve possuir um conjunto de posições candidatas para posicionar os rótulos, cada uma com sua padronização cartográfica, indicando opcionalmente uma posição preferencial. Em Christensen *et al.* [22], foi proposto um padrão cartográfico com um conjunto de oito posições

cartográficas para cada ponto, representado na Figura 3.1. O ponto central da figura representa um local (*e.g.*, uma cidade), e os números contidos em cada quadrado ao redor do ponto representam as possibilidades para colocação do rótulo (*e.g.*, nome da cidade). Quando a escolha pela rotulação de dois pontos diferentes é feita em posições candidatas que possuam sobreposição entre si, gera-se um conflito, que está indicado pelo fundo cinza na representação à direita da Figura 3.1.

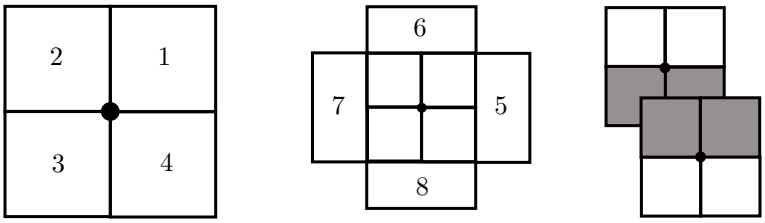

Figura 3.1: Posições Candidatas

A Figura 3.2 ilustra o mapa do Estado do Rio de Janeiro, com seus municípios e principais rodovias federais. Por ser um dos menores estados em extensão territorial e possuir mais de 90 municípios, inserir rótulos em suas cidades de maneira que a leitura do mapa seja clara é uma tarefa árdua. Observa-se, em diversos pontos, que o entendimento do mapa fica prejudicado pelas sobreposições de rótulos.

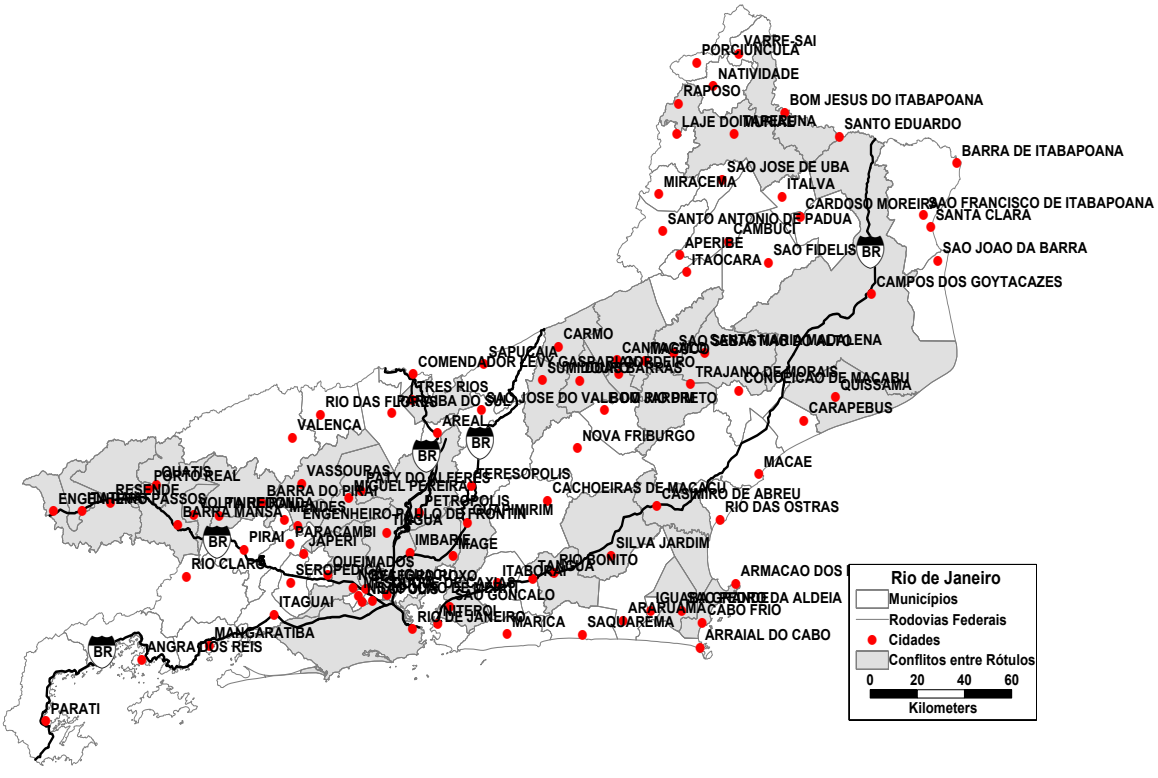

Figura 3.2: Mapa do Estado do Rio de Janeiro

O PRCP é um problema de otimização combinatória $\mathcal{NP}$-difícil [66] que, como mencionado anteriormente, consiste em atribuir rótulos a pontos específicos de uma região a ser rotulada, de modo que as sobreposições entre rótulos sejam minimizadas ou evitadas. Na literatura, é possível encontrar o PRCP com três funções objetivo distintas.

Na primeira versão, o PRCP pode ser modelado como o problema do conjunto independente máximo [110] e, nesse caso, deve-se obter a rotulação com o maior número de rótulos sem conflitos, considerando que pontos com sobreposição não devem ser rotulados. Na segunda versão, o objetivo é minimizar o número de conflitos quando todos os pontos são rotulados [89]. A última versão, que será investigada neste trabalho, busca encontrar a rotulação com a maior quantidade de rótulos sem conflitos (livres), exigindo que todos os pontos possuam rótulos [22]. Um modelo 0-1 de programação linear inteira do PRCP apresentado por Mauri *et al.* [70] para essa última versão é descrito a seguir.

Para cada ponto $i \in \{1, \ldots, N\}$, P_i representa o conjunto de posições candidatas associadas a i, e N indica o número total de pontos a serem rotulados. Seja x_{ij} uma variável binária que indica se a posição candidata j do ponto i foi escolhida ou não, $j \in P_i$. Cada posição candidata x_{ij} tem um custo associado w_{ij} que, assim como nos outros trabalhos da literatura, é igual a um em todas as instâncias. Seja S_{ij} um conjunto que representa os pares $(t, u) : t \neq i$ que contém todas as posições candidatas que representem potenciais conflitos com a posição candidata x_{ij}. A variável binária z_i indica quais pontos estão sobrepostos. Assim, a formulação é composta a seguir.

$$max \sum_{i=1}^{N} \sum_{j \in P_i} w_{ij} x_{ij} - \sum_{i=1}^{N} z_i \tag{3.1}$$

sujeito a:

$$\sum_{j \in P_i} x_{ij} = 1, \qquad \forall i = 1, \ldots, N \tag{3.2}$$

$$x_{ij} + x_{tu} - z_i \leq 1, \qquad \forall i = 1, \ldots, N; \forall j \in P_i; i \neq t; (t, u) \in S_{ij} \tag{3.3}$$

$$x_{ij}, z_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, N; \forall j \in P_i; \tag{3.4}$$

As Restrições (3.2) garantem que somente uma posição candidata por ponto será selecionada, *i.e.*, uma das posições candidatas do ponto i deve ser igual a um. As Restrições (3.3) asseguram a associação correta com as variáveis z_i quando um conflito entre os pontos i e t ocorre. As Restrições (3.4) definem o domínio das variáveis, e a função objetivo é definida na Equação (3.1), a qual busca maximizar a quantidade de rótulos sem conflito.

Marks e Shieber [66] provaram que o PRCP é $\mathcal{NP}$-difícil ao reduzir de maneira polinomial o PRCP ao problema Planar 3-SAT. Na sequência, um dos primeiros a desenvolver algoritmos heurísticos para o PRCP foi Christensen *et al.* [22], apresentando duas estratégias: a primeira, baseada no método de gradiente descendente na forma discreta, e a segunda, baseada em *Simulated Annealing*.

Verner *et al.* [106] apresentaram um algoritmo genético com máscaras que estimulavam a troca de posições candidatas sobrepostas durante a operação de *crossover*. Yamamoto *et al.* [109] desenvolveram um método heurístico baseado em busca tabu que iterativamente realizava trocas entre posições candidatas de acordo com uma lista de candidatos. As trocas respeitavam a lista tabu de proibições, com tamanho variável durante o algoritmo, e quando todos os candidatos gerados estavam na lista tabu, um critério de aspiração era adotado para escolher os candidatos mais antigos da lista.

Uma nova modelagem matemática para o problema foi proposta em Ribeiro e Lorena [89], assim como heurísticas baseadas em relaxações lagrangeanas dessa nova formulação. Cravo *et al.* [24] desenvolveram uma heurística baseada em GRASP para resolver o PRCP. Nos trabalhos de Ribeiro *et al.* [90, 91], foram apresentadas, respectivamente, uma relaxação lagrangeana com *clusters* e uma geração de colunas, conseguindo provar a otimalidade das soluções em algumas instâncias e alcançar as melhores soluções para outras.

Alvim e Taillard [8] propuseram uma heurística baseada em POPMUSIC[1], uma metaheurística geralmente aplicada a problemas cuja subdivisão em problemas menores pode ser empregada. Essa heurística divide então cada instância do PRCP em subproblemas e utiliza uma busca tabu (baseada na heurística de Yamamoto *et al.* [109]) para resolver cada um dos subproblemas. A heurística POPMUSIC obteve resultados superiores e, além de melhorar as soluções dos problemas testes utilizados até então, também foi eficiente ao resolver as novas instâncias com 13206 pontos a serem rotulados que foram introduzidas no referido trabalho.

[1] Acrônimo para *Partial Optimization Metaheuristic under Special Intensification Conditions*

Mauri *et al.* [70] investigaram um novo modelo matemático para o problema e também desenvolveram uma heurística baseada em decomposição lagrangeana. No referido trabalho, foram reportadas novas melhores soluções e valores ótimos para instâncias com até 1000 pontos.

Recentemente, Rabello *et al.* [85] propuseram uma heurística baseada em *Clustering Search* para resolver o problema de rotulação cartográfica de pontos. O algoritmo proposto combina o processo de agrupamento do *Clustering Search*, que funciona como um mecanismo de intensificação executando buscas locais em regiões promissoras, com uma componente baseada em *Simulated Annealing*, que tem como finalidade gerar novas soluções do PRCP. A heurística apresentou bom desempenho nas instâncias da literatura, encontrando todos os ótimos das instâncias pequenas e, nas instâncias maiores, conseguiu encontrar soluções melhores das que tinham sido reportadas até aquele momento.

A heurística híbrida apresentada por Rabello *et al.* [85] foi escolhida como base da proposta híbrida com mineração de dados do presente trabalho por ser uma estratégia estado-da-arte para o PRCP, e pelo desafio de introduzir a técnica de MD em uma heurística com estrutura diferente das que já foram combinadas com MD anteriormente na literatura, como detalhado no Capítulo 2. Na próxima seção, as principais características da heurística CS serão descritas.

3.2 Heurística Clustering Search para o PRCP

A heurística híbrida apresentada por Rabello *et al.* [85] combina, como mencionado anteriormente, componentes das metaheurísticas CS e SA para resolver o PRCP. Essa heurística tem um formato iterativo em que, para cada iteração do CS, é feita uma execução completa de uma heurística SA. A solução encontrada após a execução do SA é submetida ao procedimento de agrupamento da metaheurística *Clustering Search*. Por sua vez, a componente baseada em SA possui ainda dois outros laços: um de temperatura, em que acontece o resfriamento, e um laço mais interno, onde ocorrem as aplicações dos movimentos de vizinhança. No CS de Rabello *et al.* [85], após cada resfriamento completo, a temperatura é reaquecida e um novo resfriamento é executado até que a condição de parada do algoritmo seja satisfeita. A solução inicial do problema é construída aleatoriamente para ser o ponto de partida do algoritmo, escolhendo, para cada ponto, uma posição candidata aleatória para receber o rótulo daquele ponto.

Assim, a cada execução do SA, diversos movimentos da vizinhança $N(s)$ são aplicados

na solução corrente a fim de gerar novas soluções. Essas soluções são então submetidas a um agrupamento por similaridade em *clusters*, em que um procedimento de busca local é aplicado sempre que um agrupamento é considerado promissor. Um *cluster* é considerado promissor quando uma predeterminada quantidade de soluções forem a ele associadas. Após o processo de agrupamento e busca, atualiza-se o valor de temperatura, e a nova solução corrente é então ressubmetida às movimentações. Esse processo é repetido até que a temperatura atinja a temperatura de congelamento.

No SA, um movimento da vizinhança $N(s)$ é aplicado à solução atual s a fim de gerar uma nova solução s'. Cada movimento é assim definido: após a escolha de um ponto i, seu rótulo é aleatoriamente realocado para uma outra posição candidata (*e.g.*, movendo x_{i1} para x_{i3}). A nova solução s' é então submetida ao critério de aceitação probabilístico mencionado anteriormente e, assim, a busca continua. Após uma certa quantidade de movimentos, a etapa de agrupamento do CS é iniciada.

Na etapa de agrupamento, as soluções geradas após o laço interno do SA são armazenadas em *clusters*. Formalmente, os *clusters* são caracterizados por uma tripla $(\varsigma_k, \tau_k, \beta_k)$, cujos elementos são, respectivamente, o centro do *cluster*, seu volume e um indicador de ineficiência. O centro do *cluster* ς_k é a melhor solução do *cluster* k, e o representa. O volume τ_k define a quantidade de soluções que estão associadas ao *cluster* k, e o indicador de ineficiência β_k representa em quantas iterações a busca local foi aplicada ao centro do *cluster* sem obter melhorias.

No trabalho de Rabello *et al.*, cada solução gerada é incluída no *cluster* mais similar de acordo com a distância de Hamming [46], que contabiliza quantas posições candidatas diferentes s e ς_k possuem. Cada *cluster* é preenchido com soluções até que um limiar τ_{max} seja atingido. Nesse momento, esse *cluster* indica um espaço promissor de busca e, então, um procedimento de busca local é aplicado à solução central. A busca local é realizada da seguinte maneira: para cada ponto, a sua posição candidata é descartada e, para o seu lugar, escolhe-se a melhor posição candidata possível. Esse procedimento é repetido até que nenhuma melhoria seja encontrada.

A heurística híbrida CS para o PRCP apresentada em Rabello *et al.* [85] é descrita pelo Algoritmo 9. Inicialmente, uma solução aleatória é construída e, em seguida, cada um dos γ *clusters* é inicializado com uma solução central (também gerada de maneira aleatória) (linhas 2 e 3), sendo γ um parâmetro do CS. Na sequência, a solução inicial passa a ser a solução corrente s do algoritmo, que é então submetida à componente responsável pela geração de soluções, baseada em SA. A cada iteração entre as linhas

Algoritmo 9 *Clustering Search* para o PRCP

1: **CS** (γ, τ_{max}, β_{max}, T_0, T_c, ω, SA_{max})
2: Criar γ *clusters* e suas soluções centrais ς_k;
3: $s \leftarrow$ SoluçãoInicialAleatória(); $s^* \leftarrow s$;
4: **enquanto** critério de parada não satisfeito **faça**
5: $T \leftarrow T_0$;
6: **enquanto** $T > T_c$ **faça**
7: $iter \leftarrow 0$;
8: **enquanto** $iter < SA_{max}$ **faça**
9: $iter \leftarrow iter + 1$;
10: $s' \leftarrow N(s)$;
11: **se** $f(s') > f(s)$ **então**
12: $s \leftarrow s'$;
13: **senão**
14: $s \leftarrow s'$, com probabilidade $e^{\frac{-(f(s)-f(s'))}{T}}$;
15: **fim se**
16: **fim enquanto**
17: $T \leftarrow \omega T$; $k \leftarrow \underset{k\in\{1,\ldots,\gamma\}}{\arg\min}\{H_k\}$; $\tau_k \leftarrow \tau_k + 1$; $\varsigma_k \leftarrow max(s, \varsigma_k)$;
18: **se** $\tau_k = \tau_{max}$ **então**
19: $\tau_k = 0$; $s \leftarrow$Busca_Local(ς_k);
20: **se** $f(s) = f(\varsigma_k)$ **então**
21: $\beta_k \leftarrow \beta_k + 1$;
22: **se** $\beta_k = \beta_{max}$ **então**
23: $\beta_i \leftarrow 0$; $\varsigma_k \leftarrow N(\varsigma_k)$;
24: **fim se**
25: **fim se**
26: **fim se**
27: $s^* \leftarrow max(s^*, \varsigma_k)$;
28: **fim enquanto**
29: **fim enquanto**
30: **retorne** s^*;

8 e 16, diversos movimentos da vizinhança $N(s)$ são aplicados à solução atual s, que somente serão aceitos se satisfazerem ao critério de aceitação do SA já mencionado – linhas 11 a 15. Se aceito, cada movimento $N(s)$ altera uma posição candidata de um ponto escolhido aleatoriamente. Após aplicar SA_{max} movimentos, s é associada ao *cluster* k cuja solução central ς_k tem a maior similaridade com s de acordo com a distância de Hamming (linha 17).

Após associar s ao *cluster* mais similar ς_k, seu volume τ_k é incrementado na linha 17. Se o volume atual atingir o volume máximo τ_{max}, um mecanismo de busca é aplicado ao centro do *cluster* ς_k (linha 19). Caso a busca local seja realizada mais de β_{max} vezes sem melhorias, uma perturbação é realizada em ς_k com o objetivo de escapar de ótimos locais (linha 23). A perturbação efetua, basicamente, um movimento aleatório da vizinhança

$N(\varsigma_k)$ entre posições candidatas, forçando a diversificação nesse ponto do algoritmo.

A temperatura T inicia em T_0 e é atualizada de acordo com a taxa de resfriamento ω do SA na linha 17 e, caso a solução corrente s seja melhor que a melhor solução encontrada até então s^*, atualiza-se s^* com s. Finalmente, o critério de parada do CS, como descrito em Rabello *et al.* [85], é finalizar o algoritmo se a melhor solução tiver sido atualizada há mais de dois minutos.

3.3 *Data Mining* Clustering Search para o PRCP

Como mencionado anteriormente, uma das propostas desta Tese é incorporar MD à heurística híbrida desenvolvida por Rabello *et al.* [85], que possui os melhores resultados da literatura para o PRCP até então, a fim de aprimorá-la. Um dos principais desafios deste trabalho é encontrar um ponto adequado para inserção do processo de MD e como será realizada a utilização dos padrões nesta metaheurística, levando em consideração que sua estrutura e funcionamento diferem de todas as outras hibridizações com MD aplicadas na literaturas e descritas em capítulos anteriores. Nesta tese, a heurística híbrida com MD será denominada DM-CS, e será detalhada a seguir.

Como foi descrito na Seção 3.2, a estratégia de Rabello *et al.* [85] pode ser dividida basicamente em duas etapas. Na primeira etapa, soluções são geradas a partir de movimentos aleatórios aplicados à solução atual, seguindo o critério de aceitação baseado no *Simulated Annealing* a cada movimento. Na segunda, as soluções geradas são inseridas em *clusters* para posteriormente serem exploradas pela componente *Clustering Search*.

A primeira escolha ao projetar a inserção do módulo de mineração de dados foi encontrar um local adequado para construção do conjunto elite (CE). Assim, optou-se por coletar as soluções elites ao final de cada iteração do CS, ou seja, após o término da etapa de *Clustering Search* (representado pela linha 27 do Algoritmo 9). Logo, em cada iteração, a solução corrente s é adicionada ao conjunto elite caso: (i) s seja melhor que a pior solução do conjunto elite, ou (ii) o conjunto elite não esteja completamente cheio. Em ambos os casos, somente são admitidas soluções distintas das que já estão presentes no conjunto elite.

Após construir o conjunto elite, o próximo passo é decidir quando o módulo de mineração será empregado. Na abordagem proposta neste trabalho, o procedimento de mineração é chamado sempre que o conjunto elite se torna estável, seguindo ideia similar à do *Multi Data Mining* GRASP explorado em [11, 83]. Um conjunto elite é considerado

estável quando permanece ϕ iterações sem modificação.

Durante o processo de mineração, as soluções elites são enviadas ao minerador como uma base de dados de transações, onde cada solução do CE é uma transação. Uma solução do PRCP está representada na Figura 3.3, indicando para cada ponto do problema em qual posição candidata o rótulo será posicionado. Por exemplo, o ponto 5 receberá o rótulo na posição candidata 2, i.e., $x_{52} = 1$.

Pontos	1	2	3	4	5	6	7	8	9	10
Rótulos	4	3	3	1	2	1	2	1	3	4

Figura 3.3: Representação de uma solução para o PRCP, com dez pontos e quatro posições candidatas.

Porém, como o minerador trabalha com identificadores únicos para cada elemento da solução, não é possível enviar os rótulos da maneira como estão, por causa da repetição na identificação dos rótulos (*e.g.*, os pontos 2 e 3 da Figura 3.3 usam posições candidatas que possuem o mesmo identificador, de número igual a 3). Portanto, um mapeamento deve ser realizado a fim de se criar identificadores únicos para os pontos de cada solução. Assim, para cada ponto i, a identificação do seu rótulo é calculado como $|P_i| * i + j$, onde $i \in \{1, \ldots, N\}$ e $j \in P_i$. Em seguida, o minerador é executado e os padrões são extraídos. Cada padrão representa um conjunto frequente de posições candidatas, que ocorreram juntas em pelo menos sup_{min} soluções do CE. A representação do padrão e sua utilização serão detalhadas abaixo.

Por fim, a decisão de como utilizar os padrões minerados foi definida da seguinte maneira. No laço interno do SA, os movimentos da vizinhança $N(s)$ são substituídos por uma vizinhança modificada $N(s, p)$ que utiliza o padrão p. Nessa abordagem, o padrão p é o maior padrão minerado, *i.e.*, o que possui a maior quantidade de rótulos frequentes. A diferença para a vizinhança anterior é que, enquanto $N(s)$ realiza movimentos aleatórios nessa etapa do SA, na vizinhança modificada $N(s, p)$, a cada movimento, um ponto é selecionado para ser alterado e, se esse ponto ocorrer no padrão p, sua posição candidata frequente em p é inserida na solução corrente, respeitando o critério de aceitação probabilístico do *Simulated Annealing*. Caso o ponto selecionado não ocorra no padrão, ou seu rótulo selecionado já seja igual ao que está no padrão, um movimento proveniente da vizinhança original $N(s)$ é aplicado. Dessa maneira, ao invés de realizar trocas simplesmente aleatórias nas posições candidatas, escolhem-se as posições candidatas que são frequentes

em soluções de boa qualidade do CE, representadas pelo padrão p, direcionando assim a busca no espaço de soluções.

A Figura 3.4 apresenta a estrutura da solução e também do padrão minerado, similar ao que foi apresentado na Figura 3.3. Além disso, detalha a maneira como as posições candidatas do padrão p são incorporadas à solução por meio da vizinhança $N(s, p)$, considerando um exemplo com uma pequena instância de dez pontos e quatro posições candidatas. Em um primeiro movimento, o ponto 5 é aleatoriamente escolhido e sua posição candidata frequente no padrão p será incorporada à solução s, sempre respeitando o critério de aceitação já mencionado, *i.e.*, a posição candidata x_{54} é retirada da solução e x_{53} é incluída. No próximo movimento ilustrado na Figura 3.4, o ponto 8 é selecionado e o procedimento análogo é aplicado, *i.e.*, x_{81} é retirada e x_{84} incluída. Em seguida, apresenta-se como ficou a solução final após essas duas alterações de posições candidatas. As posições que estão destacadas com fundo cinza foram alteradas para acompanhar as posições candidatas do padrão.

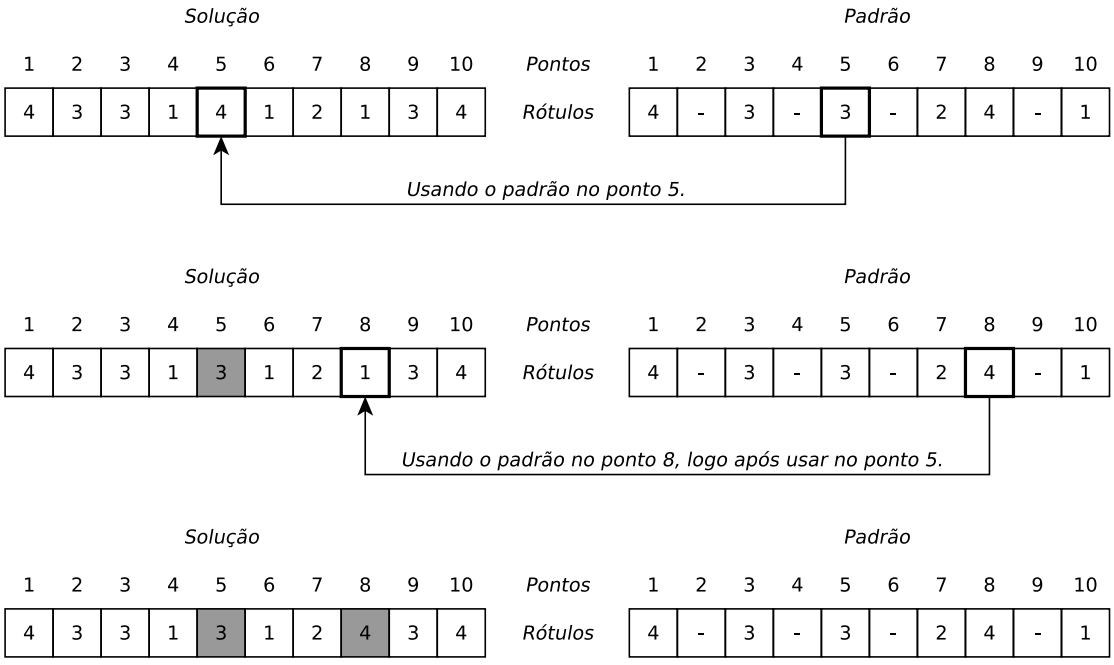

Figura 3.4: Representação de solução e do padrão minerado, e sua utilização na vizinhança $N(s, p)$.

O pseudocódigo da heurística híbrida com mineração de dados proposta (DM-CS) é detalhado no Algoritmo 10. As modificações em relação ao Algoritmo 9 estão representadas nas linhas 11–15, 32–36 e 39–42. O CE é construído nas linhas 32–36, a mineração é executada nas linhas 39–42 e a utilização dos padrões acontece nas linhas 11–15.

Algoritmo 10 Heurística Híbrida com Mineração de Dados para o PRCP

DM-CS (γ, τ_{max}, β_{max}, T_0, T_c, ω, SA_{max}, ϕ, sup_{min}, d)
Criar γ *clusters* e suas soluções centrais ς_k;
$s \leftarrow$ SoluçãoInicialAleatória(); $s^* \leftarrow s$;
$CE \leftarrow \emptyset$; $iter_{sm} \leftarrow 0$;
enquanto critério de parada não satisfeito **faça**
  $T \leftarrow T_0$;
  enquanto $T > T_c$ **faça**
    $iter \leftarrow 0$;
    enquanto $iter < SA_{max}$ **faça**
      $iter \leftarrow iter + 1$;
      se ¬ExecutouMineração() **então**
        $s' \leftarrow N(s)$;
      senão
        $s' \leftarrow N(s, p)$;
      fim se
      se $f(s') > f(s)$ **então**
        $s \leftarrow s'$;
      senão
        $s \leftarrow s'$, com probabilidade $e^{\frac{-(f(s)-f(s'))}{T}}$;
      fim se
    fim enquanto
    $T \leftarrow \omega T$; $k \leftarrow \underset{k \in \{1,\ldots,\gamma\}}{\arg\min} \{H_k\}$; $\tau_k \leftarrow \tau_k + 1$; $\varsigma_k \leftarrow max(s, \varsigma_k)$;
    se $\tau_k = \tau_{max}$ **então**
      $\tau_k = 0$; $s \leftarrow$Busca_Local(ς_k)
      se $f(s) = f(\varsigma_k)$ **então**
        $\beta_k \leftarrow \beta_k + 1$;
        se $\beta_k = \beta_{max}$ **então**
          $\beta_k \leftarrow 0$; $\varsigma_k \leftarrow N(\varsigma_k)$;
        fim se
      fim se
    fim se
    se AtualizaConjuntoEliteDeSoluções(d, s, CE) **então**
      $iter_{sm} \leftarrow 0$;
    senão
      $iter_{sm} \leftarrow iter_{sm} + 1$;
    fim se
    $s^* \leftarrow max(s^*, \varsigma_k)$;
  fim enquanto
  se $iter_{sm} \geq \phi$ **então**
    $p \leftarrow$ ExecutaMineração(CE, sup_{min});
    $iter_{sm} \leftarrow 0$;
  fim se
fim enquanto
retorne s^*;

3.4 Resultados Computacionais

A heurística *Clustering Search* (CS) de Rabello *et al.* [85] foi implementada na linguagem C++, bem como a heurística DM-CS proposta neste trabalho. Testes computacionais foram executados em um computador equipado com processador Intel®Core™ i5 CPU 650 @ 3.20GHz, com 8GB de memória RAM e Sistema Operacional Linux Ubuntu versão 14.04. Todos os experimentos foram executados em uma única *thread* e foram consideradas as instâncias do PRCP propostas em Alvim e Taillard [8] e Yamamoto *et al.* [109], com 25, 100, 250, 500, 750, 1000 e 13206 pontos e quatro posições candidatas. No último conjunto, com 13206 pontos, também foram avaliadas as instâncias com oito posições candidatas. Vale ressaltar que todo o código original do CS foi disponibilizado pelos autores e utilizado como base na implementação do DM-CS.

A Tabela 3.1 apresenta os parâmetros de ambos os algoritmos e seus valores escolhidos. A parametrização original da heurística CS [85] foi mantida. Os valores escolhidos para γ, τ_{max}, β_{max}, T_0, T_c, ω e SA_{max} são, respectivamente, 10, 7, 4, 40000, 0,01, 0,975 e 12000. Além desses, os parâmetros relativos à heurística híbrida com mineração de dados proposta neste trabalho, que são o tamanho do conjunto elite d, o valor de suporte mínimo sup_{min} e a quantidade de iterações necessárias para estabilizar o conjunto elite ϕ foram estipulados, respectivamente, em 10, 8 e 5% do total de iterações realizadas a cada resfriamento do SA. A quantidade de iterações da componente SA pode ser calculada aplicando a taxa de resfriamento ω à temperatura inicial T_0, até que se alcance a temperatura de congelamento T_c. Os parâmetros d e ϕ foram baseados na parametrização adotada por Plastino *et al.* [83], enquanto que o valor de sup_{min} foi escolhido após alguns experimentos preliminares, assegurando que o tempo de execução do minerador não se tornasse parte significativa do tempo total de execução do algoritmo. Quando utilizado suporte mínimo de valor dois, até o valor sete, o minerador consumia muito tempo computacional e a mineração se tornava custosa. Quando utilizado valores maiores que sete, o tempo despendido era muito pequeno, sem prejudicar o tamanho e a qualidade dos padrões minerados. Ambas as heurísticas foram executadas dez vezes para cada instância da literatura, com sementes diferentes.

Nas instâncias de menor porte, com 25, 100, 250, 500, 750 e 1000 pontos a serem rotulados, apenas em uma dentre as 133 instâncias o DM-CS não encontrou o melhor valor conhecido na literatura [85], ficando a 0,11% do mesmo. Nessas instâncias de pequeno porte, a inserção do módulo de mineração de dados não obteve resultados efetivos, uma vez que na grande maioria das instâncias a heurística CS original já alcançava as melhores

Tabela 3.1: Parâmetros das heurísticas CS e DM-CS

Algoritmo	Parâmetro	Significado	Valor
CS e DM-CS	γ	Número máximo de *clusters*	10
	τ_{max}	Volume máximo de cada *cluster*	7
	β_{max}	Número de tentativas antes de aplicar perturbação	4
	T_0	Temperatura inicial	40000
	T_c	Temperatura de congelamento	0,01
	ω	Taxa de resfriamento	0,975
	SA_{max}	Número de iterações do SA	12000
DM-CS	d	Tamanho do conjunto elite	10
	min_{sup}	Suporte mínimo	8
	ϕ	Iterações necessárias para o CE se estabilizar	30

soluções da literatura muito antes do módulo de mineração ser ativado. Isso acontece devido à facilidade de resolução desses problemas. No entanto, a seguir serão analisados os experimentos computacionais realizados para as instâncias de 13206 pontos.

É importante ressaltar que, embora o código do CS tenha sido disponibilizado pelos autores, o que possibilitou a realização de testes com o código original da heurística, as sementes utilizadas em seus experimentos não são conhecidas e, portanto, não foi possível realizar a reprodução exata dos experimentos reportados no artigo [85]. Dessa maneira, ainda que os resultados encontrados nas execuções do CS tenham sido próximos aos reportados no trabalho de Rabello *et al.* [85], as melhores soluções não foram exatamente as mesmas. Para facilitar a comparação, na Tabela 3.2, está reportado, para cada instância, o melhor valor conhecido (BKS), levando-se em consideração tanto os resultados alcançados em Rabello *et al.* [85], quanto os melhores resultados reportados nas novas execuções do CS, realizadas nesta Tese.

Na Tabela 3.2, estão os resultados computacionais obtidos por ambas as heurísticas para as instâncias maiores, com 13206 pontos e quatro posições candidatas. Essa tabela reporta, para cada instância e cada algoritmo, a melhor solução obtida, o valor médio de solução relativo às dez execuções e o tempo computacional médio das heurísticas CS e DM-CS. Vale ressaltar que o tempo de execução do DM-CS foi estipulado de acordo com o tempo médio despendido pela execução do CS, a fim de realizar uma comparação justa com a heurística original. Além disso, a Tabela 3.2 apresenta a diferença percentual ($\Delta_{\%} = \frac{Valor-BKS}{BKS}$, onde $Valor$ pode ser tanto o custo da melhor solução do respectivo algoritmo – $\Delta_{\%}$ da Melhor –, quanto a média de custo de solução – $\Delta_{\%}$ da Média) dessas heurísticas em relação aos BKS. Na comparação entre os algoritmos, os valores em negrito representam os melhores resultados obtidos e, ao final da tabela, encontra-se a média geral

das diferenças percentuais.

Tabela 3.2: Resultados computacionais do CS e DM-CS para instâncias com 13206 pontos e quatro posições candidatas

		CS (Rabello *et al.* [85])				**DM-CS**				
Inst.	**BKS**	**Melhor**	**Δ%**	**Média**	**Δ%**	**Melhor**	**Δ%**	**Média**	**Δ%**	**Tempo[a]**
1	12479	12478	-0,01	12472,2	-0,05	**12493**	0,11	**12487,1**	0,06	477,45
2	12113	12113	0,00	12102,0	-0,09	**12141**	0,23	**12133,7**	0,17	607,09
3	11895	11895	0,00	11882,4	-0,11	**11926**	0,26	**11914,2**	0,16	624,59
4	11702	11698	-0,03	11692,3	-0,08	**11733**	0,26	**11726,4**	0,21	680,64
5	11758	11758	0,00	11746,0	-0,10	**11808**	0,43	**11790,9**	0,28	712,92
6	10889	10889	0,00	10875,4	-0,12	**10945**	0,51	**10931,1**	0,39	716,79
7	10491	10487	-0,04	10476,1	-0,14	**10547**	0,53	**10530,6**	0,38	543,92
8	10821	10821	0,00	10802,9	-0,17	**10871**	0,46	**10859,2**	0,35	663,53
9	10806	10805	-0,01	10795,6	-0,10	**10859**	0,49	**10844,9**	0,36	628,49
10	10397	10391	-0,06	10378,5	-0,18	**10454**	0,55	**10432,4**	0,34	602,95
11	10068	10068	0,00	10051,2	-0,17	**10131**	0,63	**10110,4**	0,42	652,42
12	9955	9941	-0,14	9928,5	-0,27	**9992**	0,37	**9976,0**	0,21	529,51
13	10787	10782	-0,05	10776,3	-0,10	**10871**	0,78	**10849,8**	0,58	807,44
14	10121	10121	0,00	10102,5	-0,18	**10181**	0,59	**10168,7**	0,47	618,45
15	9699	9699	0,00	9686,6	-0,13	**9768**	0,71	**9755,2**	0,58	723,61
16	9306	9290	-0,17	9280,5	-0,27	**9368**	0,67	**9352,6**	0,50	781,07
17	10255	10250	-0,05	10236,0	-0,19	**10302**	0,46	**10277,6**	0,22	472,71
18	9509	9501	-0,08	9483,3	-0,27	**9578**	0,73	**9564,7**	0,59	644,29
19	9074	9063	-0,12	9054,3	-0,22	**9159**	0,94	**9135,9**	0,68	829,29
20	8598	8598	0,00	8585,0	-0,15	**8701**	1,20	**8685,6**	1,02	902,03
Média			-0,04		-0,15		**0,55**		**0,40**	

[a] - Tempos computacionais médios, em segundos, para ambos algoritmos

Observando a Tabela 3.2, é possível notar que, para todas as instâncias, a heurística DM-CS apresenta melhores soluções e melhores valores médios de solução, sendo que o ganho percentual geral relativo à melhor solução do DM-CS foi em média igual a 0,55%, e, relativo ao valor médio de solução, de 0,40%. Para as 20 instâncias comparadas nessa tabela, o DM-CS obteve todas as melhores soluções da literatura.

A Tabela 3.3 possui a mesma estrutura da Tabela 3.2, porém apresenta os resultados para as instâncias com 13206 pontos e oito posições candidatas. Em relação à Tabela 3.3,

Tabela 3.3: Resultados computacionais do CS e DM-CS para instâncias com 13206 pontos e oito posições candidatas

		CS (Rabello *et al.* [85])				**DM-CS**				
Inst.	**BKS**	**Melhor**	**Δ%**	**Média**	**Δ%**	**Melhor**	**Δ%**	**Média**	**Δ%**	**Tempo[a]**
1	12911	12842	-0,53	12828,0	-0,64	**12909**	-0,02	**12898,0**	-0,10	1375,70
2	12462	12351	-0,89	12338,8	-0,99	**12453**	-0,07	**12443,4**	-0,15	1368,41
3	12496	12396	-0,80	12382,2	-0,91	**12510**	0,11	**12498,0**	0,02	1453,23
4	12345	12259	-0,70	12234,8	-0,89	**12380**	0,28	**12364,1**	0,15	1278,91
5	12451	12350	-0,81	12333,9	-0,94	**12478**	0,22	**12463,2**	0,10	1194,30
6	11688	11585	-0,88	11556,7	-1,12	**11726**	0,33	**11707,7**	0,17	1278,99
7	11330	11229	-0,89	11210,1	-1,06	**11406**	0,67	**11393,0**	0,56	1115,36
8	11641	11521	-1,03	11508,8	-1,14	**11679**	0,33	**11666,6**	0,22	1598,29
9	11318	11217	-0,89	11193,1	-1,10	**11381**	0,56	**11361,6**	0,39	1358,29
10	11257	11150	-0,95	11133,2	-1,10	**11348**	0,81	**11333,6**	0,68	1131,19
11	10948	10866	-0,75	10835,4	-1,03	**11054**	0,97	**11023,6**	0,69	1444,57
12	10717	10620	-0,91	10601,2	-1,08	**10812**	0,89	**10792,9**	0,71	1486,93
13	11694	11576	-1,01	11558,7	-1,16	**11767**	0,62	**11745,2**	0,44	1424,35
14	10669	10574	-0,89	10541,1	-1,20	**10774**	0,98	**10738,3**	0,65	1481,33
15	10576	10479	-0,92	10459,4	-1,10	**10711**	1,28	**10682,1**	1,00	1221,68
16	10231	10121	-1,08	10110,2	-1,18	**10365**	1,31	**10333,2**	1,00	1345,64
17	11246	11122	-1,10	11098,2	-1,31	**11302**	0,50	**11281,9**	0,32	1370,99
18	10082	9975	-1,06	9963,1	-1,18	**10233**	1,50	**10221,2**	1,38	1216,37
19	10031	9946	-0,85	9914,3	-1,16	**10193**	1,61	**10158,2**	1,27	1152,04
20	9599	9499	-1,04	9474,9	-1,29	**9728**	1,34	**9707,6**	1,13	1352,63
Média			-0,90		-1,08		**0,71**		**0,53**	

[a] - Tempos computacionais médios, em segundos, para ambos algoritmos

a heurística DM-CS também foi capaz de apresentar soluções melhores que as reportadas na literatura, encontrando 18 novas soluções nas 20 instâncias testadas. A diferença percentual foi, em média, de 0,71% para o critério de melhor solução, e de 0,53%, quando comparados os valores de custos médios de solução.

Na sequência, a heurística proposta DM-CS é comparada com as melhores estratégias da literatura: três heurísticas baseadas em POPMUSIC [8], a heurística ILS de [80] e a heurística CS original [85]. Tal comparação, apresentada nas Tabelas 3.4 e 3.5, segue a maneira tradicional de como é feita a comparação nos artigos da literatura que abordam o PRCP. Assim, são apresentados a porcentagem de rótulos livres (%PRL) obtida pela heurística DM-CS, e a obtida pelos outros métodos já mencionados para instâncias com quatro e oito posições candidatas.

Tabela 3.4: Comparando o percentual de rótulos livres (%PRL) com outras abordagens da literatura para instâncias com 13206 pontos e quatro posições candidatas.

	Alvim e Taillard [8]			Oliveira *et al.* [80]	Rabello *et al.* [85]	DM-CS
Inst.	Tabu (50n)	Pop (asc)	Pop (10)	ILS	CS	
1	92,20	93,10	92,55	**94,65**	94,49	94,60
2	88,29	89,18	88,73	91,05	91,66	**91,94**
3	86,12	87,12	86,69	88,91	90,06	**90,31**
4	83,80	84,87	84,42	86,43	88,61	**88,85**
5	84,25	85,26	84,73	88,47	89,00	**89,41**
6	74,31	75,92	75,49	79,89	82,44	**82,88**
7	69,23	71,76	71,16	76,34	79,44	**79,87**
8	73,21	75,15	74,94	79,68	81,91	**82,32**
9	73,00	74,81	74,28	79,47	81,83	**82,23**
10	67,47	70,33	69,96	75,97	78,73	**79,16**
11	63,56	67,11	66,66	72,85	76,19	**76,72**
12	62,01	65,62	65,10	72,17	75,38	**75,66**
13	73,19	74,54	74,15	79,48	81,68	**82,32**
14	64,44	67,34	67,12	73,51	76,63	**77,09**
15	61,95	63,03	62,86	70,22	73,41	**73,97**
16	58,23	59,21	58,87	66,27	70,47	**70,94**
17	67,07	68,82	68,17	74,75	77,65	**78,01**
18	60,02	60,87	60,44	68,22	72,01	**72,53**
19	56,10	56,75	56,55	64,32	68,71	**69,35**
20	51,89	52,23	51,98	60,66	65,08	**65,89**
Média	70,52	72,15	71,74	77,17	79,77	**80,20**

Como é possível observar na Tabela 3.4, considerando o critério %PRL, a heurística proposta DM-CS claramente consegue obter os melhores resultados da literatura, com exceção de uma instância, onde o ILS reportou a melhor porcentagem de rótulos livres. Em

relação à Tabela 3.5, o DM-CS novamente encontrou melhores resultados em comparação com os outros métodos, com exceção das duas primeiras instâncias, em que o melhor valor foi reportado pelo CS [85].

Tabela 3.5: Comparando o percentual de rótulos livres (%PRL) com outras abordagens da literatura para instâncias com 13206 pontos e oito posições candidatas.

	Alvim e Taillard [8]			Rabello *et al.* [85]	DM-CS
Inst.	Tabu (50n)	Pop (asc)	Pop (10)	CS	
1	97,13	97,61	97,22	**97,77**	97,75
2	92,54	93,43	92,70	**94,37**	94,30
3	92,87	94,09	93,06	94,62	**94,73**
4	91,40	92,38	91,54	93,48	**93,75**
5	92,44	93,47	92,51	94,28	**94,49**
6	84,56	85,85	84,96	88,51	**88,79**
7	80,54	82,23	81,35	85,79	**86,37**
8	83,95	85,39	84,26	88,15	**88,44**
9	80,57	81,84	81,07	85,70	**86,18**
10	79,84	81,29	80,61	85,24	**85,93**
11	76,58	78,45	77,46	82,90	**83,70**
12	74,36	76,00	75,17	81,15	**81,87**
13	84,55	85,70	84,48	88,55	**89,10**
14	73,63	75,09	74,58	80,79	**81,58**
15	72,57	74,20	73,29	80,08	**81,11**
16	68,35	70,82	70,04	77,47	**78,49**
17	79,22	80,70	79,52	85,16	**85,58**
18	66,96	69,07	68,13	76,34	**77,49**
19	65,53	68,89	67,86	75,96	**77,18**
20	58,98	64,52	63,66	72,69	**73,66**
Média	79,83	81,55	80,67	85,45	**86,02**

Alguns experimentos adicionais foram realizados a fim de ilustrar e comparar o comportamento dos dois algoritmos analisados neste trabalho. As Figuras 3.5 e 3.6 mostram como se comportam ambos os algoritmos, reportando, por iteração, os valores de solução obtidos por cada uma das estratégias na instância 1 com 13206 pontos e, respectivamente, com quatro e oito posições candidatas. Os resfriamentos totais e os reaquecimentos acontecem a cada 600 iterações. À medida que a temperatura vai decrescendo, ambas estratégias atingem soluções melhores (trata-se de um problema de maximização). Na Figura 3.5, as minerações do DM-CS ocorreram nas iterações 138, 631, 1224, 1806 e 2431. Na Figura 3.6, as minerações acontecem nas iterações 168, 633, 1235 e 1811.

Nas duas figuras, com exceção da primeira mineração, em todas as outras o custo de solução do DM-CS melhorou consideravelmente em relação ao CS logo após a realização da primeira mineração. Esse comportamento corrobora com a hipótese de que os padrões

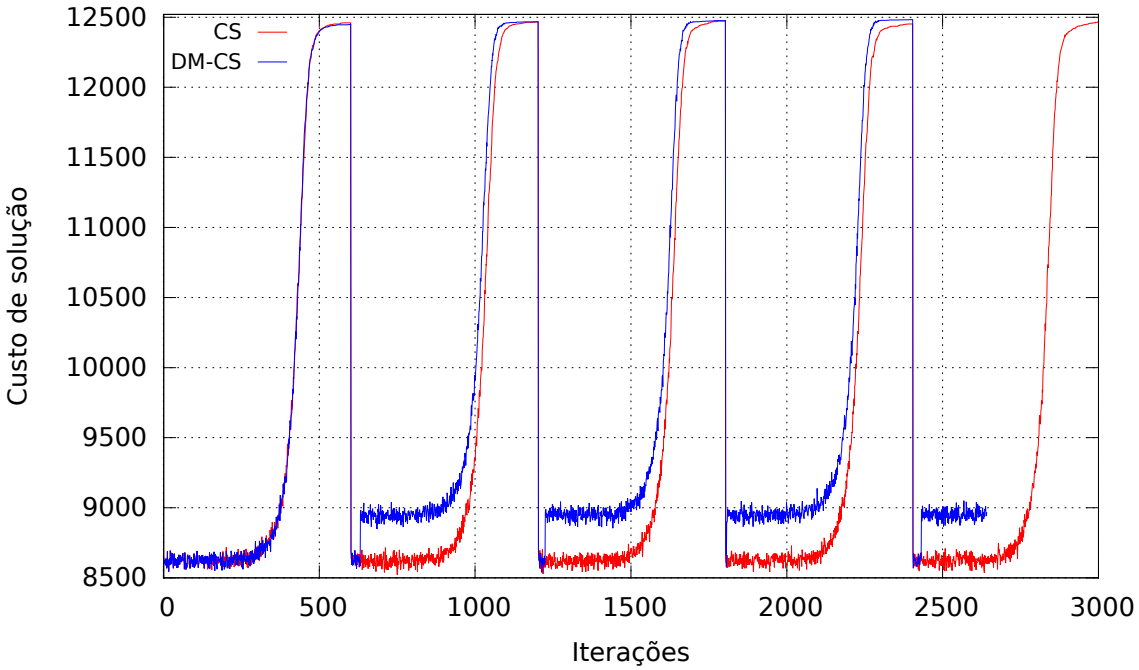

Figura 3.5: Análise de comportamento das heurísticas CS e DM-CS para a instância 1, com 13206 pontos e quatro posições candidatas.

extraídos auxiliam na busca por melhores soluções, atuando como uma componente de intensificação. Vale ressaltar que a execução do DM-CS pode finalizar com menos iterações (ou mais) que o CS. Isso ocorre pois não há como garantir que cada iteração terá exatamente o mesmo tempo em ambas as heurísticas, nem que o último resfriamento será completo na heurística DM-CS, devido ao tempo de execução que foi limitado ao mesmo tempo de execução do CS.

As Figuras 3.7 e 3.8 representam outra comparação entre os dois algoritmos abordados, baseada nos gráficos *Time-to-Target Plots* (`tttplots`) [7], que são usados para analisar o comportamento de algoritmos com componentes aleatórias. Esses gráficos mostram a probabilidade acumulada, no eixo das ordenadas, de um algoritmo encontrar uma solução melhor ou igual a uma solução alvo prefixada, em um tempo de execução definido no eixo das abscissas. Nesses experimentos, foram realizadas 100 execuções de ambas estratégias.

É possível notar que o comportamento da versão híbrida supera o CS original, em ambos os casos. Na Figura 3.7, por exemplo, a probabilidade do DM-CS encontrar a solução alvo em 500 segundos é de quase 100% enquanto que para o CS essa mesma probabilidade está em torno de 45%.

Outra verificação feita foi baseada na ferramenta `tttplots-compare`, um método desenvolvido por Ribeiro e Rosseti [87]. Esse método calcula a probabilidade $Pr(A_1 \leq A_2)$

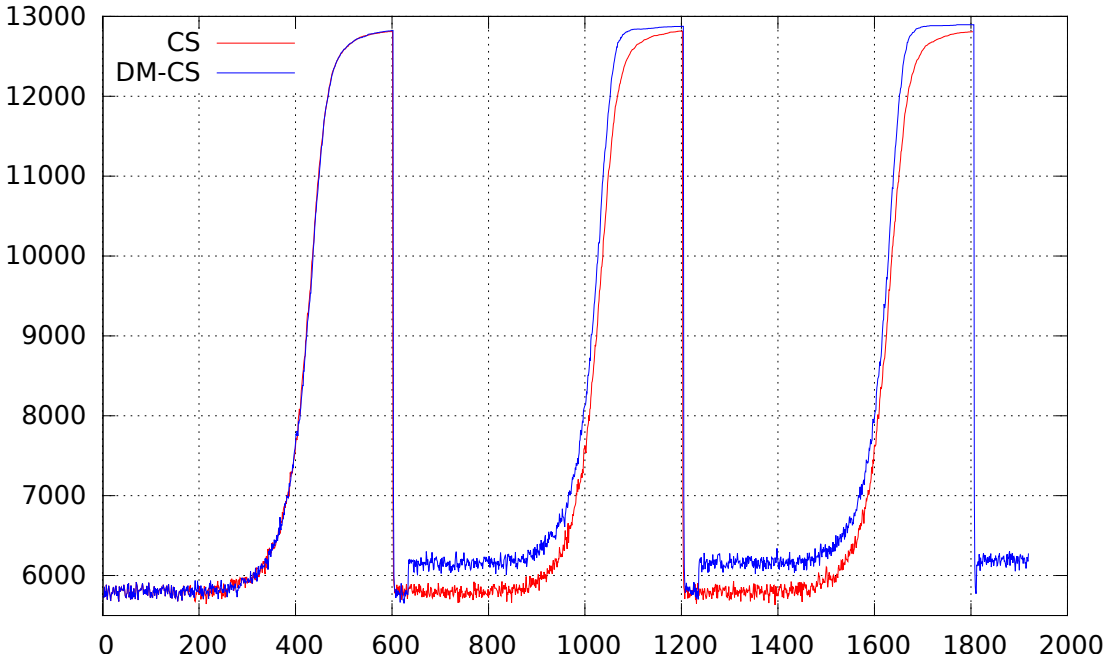

Figura 3.6: Análise de comportamento das heurísticas CS e DM-CS para a instância 1, com 13206 pontos e oito posições candidatas.

de um algoritmo A_1 encontrar uma solução de qualidade tão boa quanto uma solução de custo prefixado consumindo menos tempo computacional que o algoritmo A_2. No caso do presente trabalho, o algoritmo A_1 identifica a heurística DM-CS, enquanto que A_2 representa a heurística CS. Ambas as heurísticas foram comparadas utilizando as mesmas instâncias e soluções alvos que os testes anteriores apresentados para o `tttplots` nas Figuras 3.7 and 3.8. Para a primeira instância, o valor calculado $Pr(A_1 \leq A_2) = 88,95\%$ indica que o DM-CS possui uma probabilidade alta de encontrar a solução alvo antes que a heurística CS. Para a segunda instância, a probabilidade calculada foi de $Pr(A_1 \leq A_2) = 81,27\%$, evidenciando mais uma vez a superioridade da heurística proposta que incorpora o módulo de mineração de dados.

A última análise realizada foi verificar se os ganhos referentes à média de solução reportados nas Tabelas 3.2 e 3.3 têm significância estatística, realizando o teste estatístico não paramétrico de *Wilcoxon* pareado e unicaudal [101]. Esse teste é geralmente aplicado para avaliar dois algoritmos que possuem componentes aleatórias, e identificar se a diferença entre as médias de resultados obtidos por esses algoritmos foi, de fato, devido à superioridade de algum deles ou simplesmente devido à aleatoriedade dos métodos [101]. Para realizar o teste, foram considerados p-valor igual a $0,01$.

A Tabela 3.6 mostra a comparação entre os algoritmos CS e DM-CS, separadas por

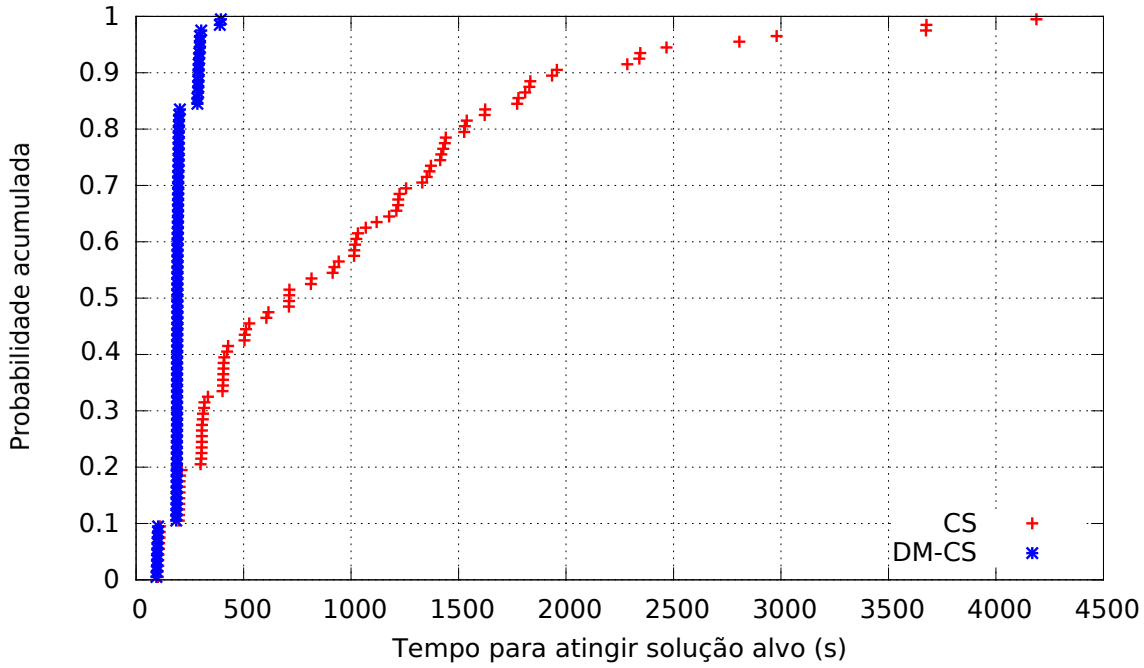

Figura 3.7: Gráfico *time-to-target* para instância 1 com 13206 pontos e quatro posições candidatas, com o valor de solução alvo igual a 12472.

grupos de instâncias de quatro e oito posições candidatas. O número fora dos parênteses indica em quantas instâncias aquela estratégia foi melhor que a outra em relação à média de solução, enquanto que o valor entre parênteses indica o número de vezes em que houve significância estatística para o referido resultado, *i.e.*, sendo o p-valor menor que $0,01$ e, consequentemente, indicando que a probabilidade de a diferença de desempenho entre os algoritmos ser devido à aleatoriedade é menor que 1%.

Tabela 3.6: Análise de significância estatística

Algoritmos	Grupo de instâncias	
	13206_4	13206_8
CS	0 (0)	0 (0)
DM-CS	20 (20)	20 (20)

Ao analisar todos os experimentos realizados, os resultados evidenciaram a efetividade da introdução de mineração de dados na heurística original, alcançando melhores soluções, se comparadas com as da heurística original, assim como com as soluções da literatura. O ganho médio em relação às melhores soluções conhecidas, considerando as instâncias com 13206 pontos e quatro posições candidatas, foi de 0,55% e para os valores médios de solução esse ganho foi de 0,40%. Para as instâncias com 13206 pontos e oito posições candidatas, o ganho médio em relação às melhores soluções conhecidas foi de $0,71\%$, enquanto que em

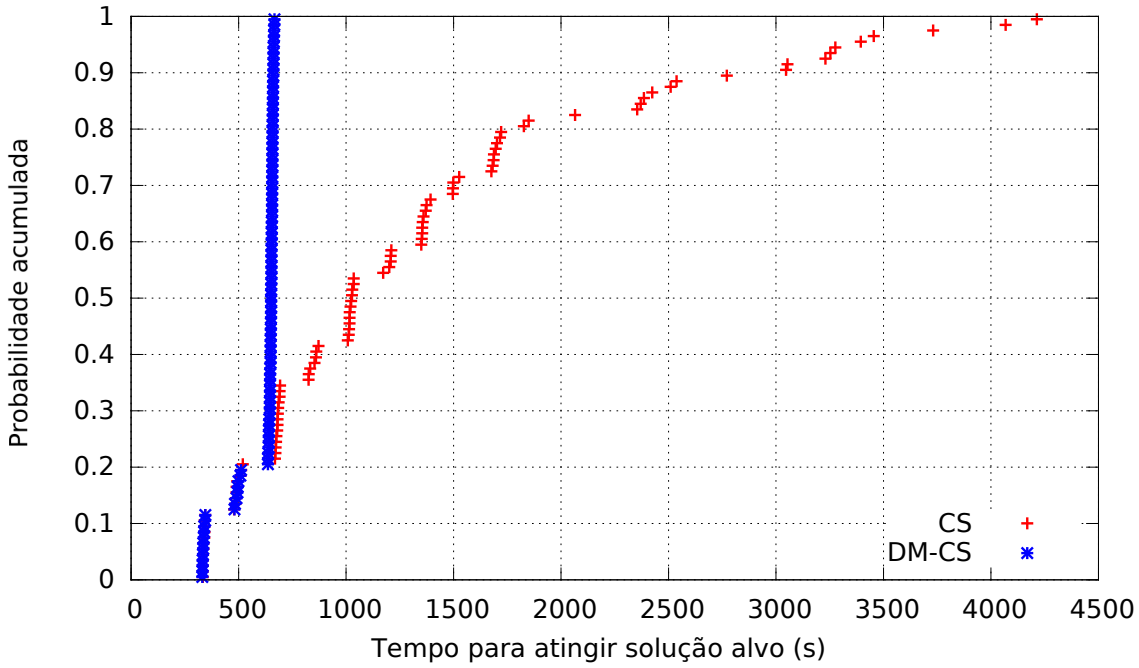

Figura 3.8: Gráfico *time-to-target* para instância 1 com 13206 pontos e oito posições candidatas, com o valor de solução alvo igual a 12828.

relação às médias de solução foi de $0,53\%$. Além disso, experimentos complementares, realizados com o `tttplots` e o `tttplots-compare`, comprovaram a superioridade da versão híbrida com mineração no que diz respeito à convergência do método. Essas comparações, validadas com uma análise estatística, demonstraram o desempenho do método proposto.

No próximo capítulo, será descrita a proposta de inserção de mineração de dados para uma heurística baseada em *Simulated Annealing* para solucionar o Problema de Agrupamento Capacitado com Centro Geométrico.

Capítulo 4

Problema de Agrupamento Capacitado com Centro Geométrico

4.1 Introdução

Clusterização é uma tarefa da área de Análise de Dados que consiste em agrupar um conjunto de objetos em classes ou *clusters* de maneira que objetos em um mesmo *cluster* sejam similares entre si e dissimilares em relação a objetos dos demais *clusters* [47]. Essa tarefa possui aplicações em diversas áreas, tais como processamento de imagens, análise de mercado, sistemas de recomendação, medicina, biologia, entre várias outras [47].

A Clusterização é também tratada como um problema de otimização combinatória [55, 72]. Sejam I um conjunto de objetos a serem agrupados e J um conjunto de *clusters*. Considere d_{ij} a medida de dissimilaridade entre o objeto i e o centro do *cluster* j, que pode ser calculada pela distância Euclidiana, e p como o número predefinido de *clusters* a serem considerados. Sejam x_{ij} uma variável binária que indica se o elemento i está associado ao *cluster* j e y_j uma variável binária que indica se o ponto j é mediana do *cluster* $j \in J$. A formulação do problema de Clusterização é a que segue.

$$min \sum_{i \in I} \sum_{j \in J} d_{ij} x_{ij} \tag{4.1}$$

sujeito a:

$$\sum_{j \in J} y_j = p \tag{4.2}$$

$$\sum_{j \in J} x_{ij} = 1 \quad i \in I \tag{4.3}$$

$$x_{ij} \leq y_j \quad i \in I, j \in J \tag{4.4}$$

$$x_{ij}, y_j \in \{0, 1\}, \quad i \in I, j \in J \tag{4.5}$$

A Restrição 4.2 define que exatamente p *clusters* sejam criados e a Restrição 4.3 garante que cada objeto i seja associado a exatamente um *cluster* j. A Restrição 4.4 assegura que pontos sejam associados somente a *clusters* existentes e, por fim, a Restrição 4.5 define o domínio das variáveis de decisão.

No problema de Agrupamento Capacitado (*Capacitated Clustering Problem* – CCP), cada elemento i possui um peso q_i associado e cada *cluster* j possui uma capacidade máxima Q_j que deve ser respeitada. Nesse caso, a Restrição 4.6 deve ser adicionada ao modelo apresentado nas Equações 4.1 a 4.5:

$$\sum_{i \in I} q_i x_{ij} \leq Q_j \quad j \in J \tag{4.6}$$

No CCP, o centro de um *cluster* é a mediana (um objeto $i \in I$) que minimiza as dissimilaridades entre os elementos que pertencem a esse *cluster*. Nesta Tese, é investigada uma variante do CCP denominada problema de agrupamento capacitado com centro geométrico (*Capacitated Centred Clustering Problem* – CCCP), proposta por Negreiros e Palhano [75]. No CCCP, o objetivo continua a ser encontrar o melhor agrupamento dos objetos em *clusters*, de maneira que seja minimizada a dissimilaridade entre objetos de mesmos *clusters*. A diferença é que, no CCCP, os centros dos *clusters* passam a ser definidos pelo centro geométrico de cada agrupamento (também denominado centroide), em vez das medianas. As principais aplicações reais para o CCCP estão no projeto de zonas de coleta de lixo e localização de plataformas de exploração de petróleo em alto mar, mas diversas outras podem ser encontradas em [56, 75].

A seguinte formulação para o CCCP foi proposta por Negreiros e Palhano [75].

$$min \sum_{i \in I} \sum_{j \in J} ||a_i - \overline{y_j}||^2 x_{ij} \tag{4.7}$$

sujeito a:

$$\sum_{j \in J} x_{ij} = 1 \quad i \in I \tag{4.8}$$

$$\sum_{i \in I} x_{ij} = n_j \quad j \in J \tag{4.9}$$

$$\sum_{i \in I} a_i x_{ij} = n_j \overline{y_j} \quad j \in J \tag{4.10}$$

$$\sum_{i \in I} q_i x_{ij} \leq Q_j \quad j \in J \tag{4.11}$$

$$\overline{y_j} \in \mathbb{R}^l, n_j \in N, x_{ij} \in \{0,1\}, \quad i \in I, j \in J \tag{4.12}$$

onde $\overline{y_j}$ é o centroide do *cluster* j, a variável n_j é a quantidade de elementos no *cluster* j, o valor a_i a posição do elemento i no espaço $\mathbb{R}^l$, e $|J| = p$. A função objetivo 4.7 minimiza a soma das dissimilaridades, isto é, as distâncias de cada ponto a_i ao centroide de seu *cluster* $\overline{y_j}$. A Restrição 4.9 contabiliza a quantidade de elementos por *cluster* e a Restrição 4.10 posiciona o centroide do *cluster* $\overline{y_j}$ em seu centro geométrico, dado pela média das coordenadas dos pontos que a ele pertencem. As Restrições 4.8 e 4.11 são iguais às Restrições 4.3 e 4.6, respectivamente. No CCCP, o fato de o centroide do *cluster* não ser necessariamente um elemento de I faz com que o modelo seja não linear, tanto na função objetivo 4.7, quanto na Restrição 4.10.

A Figura 4.1 ilustra uma instância do CCCP com vinte pontos e três *clusters*. Cada estrela representa um ponto e cada aresta representa a conexão do ponto ao centroide do seu respectivo *cluster*.

Na literatura, alguns trabalhos abordaram diretamente o CCCP. O problema foi introduzido por Negreiros and Palhano [75] e, além da modelagem, os autores também apresentaram uma heurística em duas fases. A primeira, baseia-se no algoritmo de Forgy [34] para construir uma solução inicial, e a segunda é baseada na metaheurística *Variable Neighborhood Search* (VNS) [71]. Instâncias do *benchmark* do problema CCP foram usadas como problemas-teste a fim de avaliar o algoritmo descrito em [75].

Pereira e Senne [81] propuseram um método de geração de colunas, adaptando uma versão originalmente proposta para o problema das p-medianas capacitado, para resolver

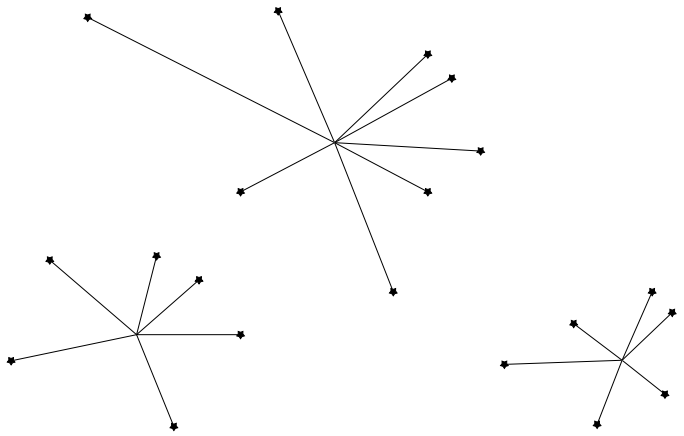

Figura 4.1: Representação de uma instância do CCCP.

o CCCP. O algoritmo faz uso de uma relaxação langrangeana/surrogate para estabilizar o processo de geração de colunas. Em grande parte do conjunto de instâncias, novas soluções foram reportadas.

Chaves e Lorena [20] apresentaram uma heurística híbrida baseada nas metaheurísticas *Clustering Search* (CS) e *Simulated Annealing* (SA). O SA é empregado como uma componente que gera soluções para serem inseridas no processo de agrupamento do CS – assim como na heurística descrita no Capítulo 2 para o problema de rotulação cartográfica de pontos – que poderão ser exploradas pelo módulo de busca local. A avaliação das soluções é feita de maneira aproximada em alguns pontos do algoritmo, visando diminuir o alto custo computacional de sempre recalcular os centroides a cada nova solução encontrada. A heurística CS-SA obteve bons resultados tanto nas instâncias de pequeno porte, quanto nas instâncias maiores, reportando várias novas melhores soluções para o problema.

Em Chaves e Lorena [21], os mesmos autores propuseram uma nova heurística híbrida baseada em CS e Algoritmos Genéticos (GA), em que o GA é então usado como gerador de soluções para o CS. Essa nova heurística mostrou-se competitiva para o CCCP, conseguindo alcançar as melhores soluções em 18 das 25 das instâncias testadas, sendo 11 delas novas soluções encontradas.

A heurística híbrida CS-SA apresentada por Chaves e Lorena [20] foi escolhida como base da proposta híbrida com mineração de dados do presente trabalho por ser uma das estratégias estado-da-arte para o CCCP, e por apresentar uma heurística baseada em *Simulated Annealing* e *Clustering Search*.

4.2 Heurística CS-SA para o CCCP

A heurística CS-SA para CCCP apresentada por Chaves e Lorena [20] segue uma estrutura bem similar à heurística CS de Rabello *et al.* [85], apresentada no Capítulo 3 para o PRCP. No entanto, antes de detalhá-la, é importante evidenciar um detalhe na representação da solução do CCCP que influencia diretamente na descrição das componentes do algoritmo. Como a avaliação da função objetivo para o CCCP tem um alto custo computacional, uma vez que os centroides são desconhecidos e devem ser recalculados em todas as reavaliações de novas soluções, os autores optaram por realizar um cálculo aproximado da função objetivo, utilizando uma estrutura computacional que já havia sido empregada para o problema *Capacitated p-Median Problem* (CPMP) [19].

Nessa representação, ao invés de utilizar o centroide como ponto de referência para o cálculo das distâncias, um dos pontos pertencentes ao agrupamento passa a ser o ponto de referência. Dessa maneira, utilizando um ponto existente, evita-se recalcular o centroide, e o cálculo da função objetivo pode ser acelerado uma vez que todas as distâncias entre pontos podem ser calculadas previamente (*e.g.*, calculando a matriz de distâncias em um pré-processamento). Caso contrário, se os centroides fossem recalculados a cada alteração de solução, os tempos computacionais se tornariam proibitivos. Para o cálculo aproximado, esses pontos de referência são denominados medianas (uma referência herdada do trabalho de Chaves *et al.* [19], que realmente trabalhava com medianas). Assim, o cálculo simplificado do custo da solução é feito de maneira a somar as distâncias – já conhecidas – dos pontos de cada agrupamento à sua respectiva mediana. A Figura 4.2 apresenta a estrutura de uma solução, onde estão representados os dez pontos que caracterizam o problema e suas respectivas medianas 3, 5 e 7 associadas. Pela figura, é possível perceber que os pontos 2, 3 e 9 pertencem ao mesmo agrupamento (associados à mediana 3), assim como 1, 5 e 8 (associados à mediana 5), e finalmente os pontos 4, 6, 7 e 10 (associados à mediana 7).

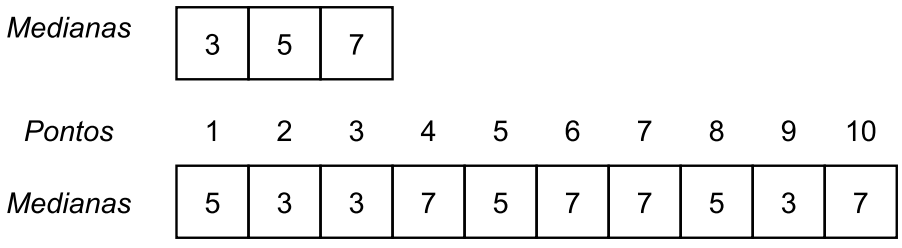

Figura 4.2: Representação de uma solução do CCCP.

A heurística baseada em CS e SA de Chaves e Lorena [20] é iniciada com uma solução aleatória, escolhendo pontos como medianas de cada um dos agrupamentos[1] de maneira arbitrária. Em seguida, para cada ponto diferente das medianas e ainda não agrupado, é feita sua associação ao agrupamento de mediana mais próxima, de acordo com a distância euclidiana para as medianas, de modo que a capacidade do agrupamento não seja excedida.

Após a construção de uma solução inicial, a estrutura do SA é ativada com a aplicação de movimentos que modificam a solução corrente. Para cada valor de temperatura, uma vizinhança N^k (de um conjunto de vizinhanças que contém cinco tipos de movimentos, detalhados na seção a seguir) é selecionada aleatoriamente para ser aplicada à solução corrente. A solução gerada a partir de N^k é então submetida ao critério de aceitação probabilístico do SA. Caso a solução gerada por esse movimento obtenha um custo melhor, ela é aceita. Caso contrário, existe uma probabilidade de aceitação que é baseada no valor da temperatura atual e também no valor da diferença entre o custo da solução base e a solução encontrada após o movimento. A quantidade de movimentos que são aplicados à solução corrente s é um parâmetro de entrada da estratégia, assim como as temperaturas inicial e final, e a taxa de resfriamento da temperatura.

Cinco vizinhanças diferentes foram definidas no trabalho de Chaves e Lorena [20], denominadas N^1, N^2, N^3, N^4 e N^5. A seguir, cada uma dessas vizinhanças são detalhadas separadamente. Nas Figuras 4.3 a 4.6, cada círculo indica um agrupamento, cada triângulo representa uma mediana do agrupamento e as setas demonstram a direção do movimento realizado.

4.2.1 Vizinhança N^1

Um movimento de N^1 é obtido ao trocar a alocação de dois pontos de diferentes agrupamentos.

[1] Apesar de sinônimos, os termos agrupamento e *cluster* no contexto deste trabalho serão relacionados a conceitos diferentes. Para padronizar, o termo agrupamento estará relacionado a um conjunto de elementos do problema de otimização CCCP, enquanto que o termo *cluster* será empregado em referência a um conjunto de soluções que são similares de acordo com a metaheurística CS.

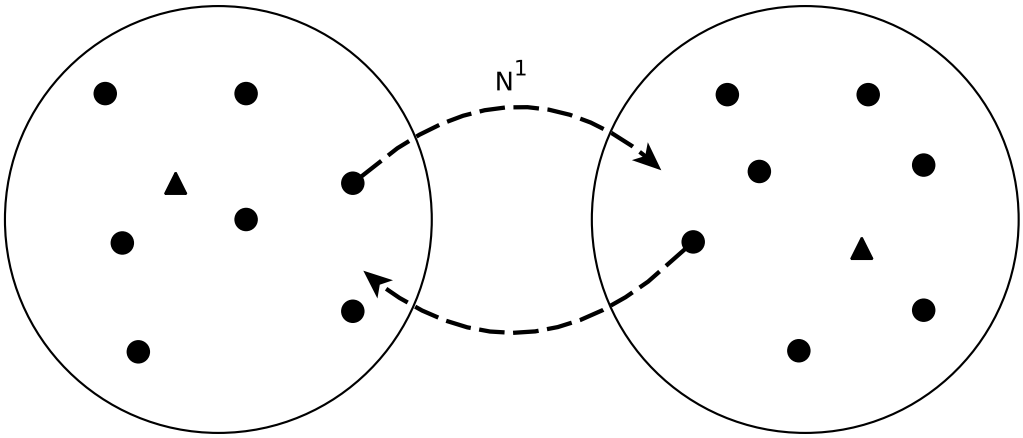

Figura 4.3: Representação de uma aplicação de um movimento de N^1.

4.2.2 Vizinhança N^2

Um movimento de N^2 troca uma mediana por um ponto que está associado a ela. Vale ressaltar que, embora esse movimento não acarrete em uma mudança no custo original da solução (calculado à partir do centroide), o cálculo do custo simplificado é alterado e isso pode modificar a busca por novas soluções.

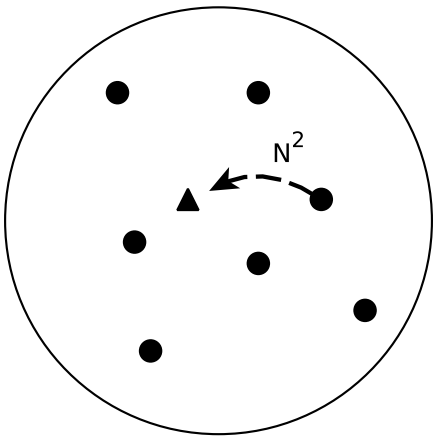

Figura 4.4: Representação de uma aplicação de um movimento de N^2.

4.2.3 Vizinhança N^3

Um movimento de N^3 move um ponto de um agrupamento para outro qualquer.

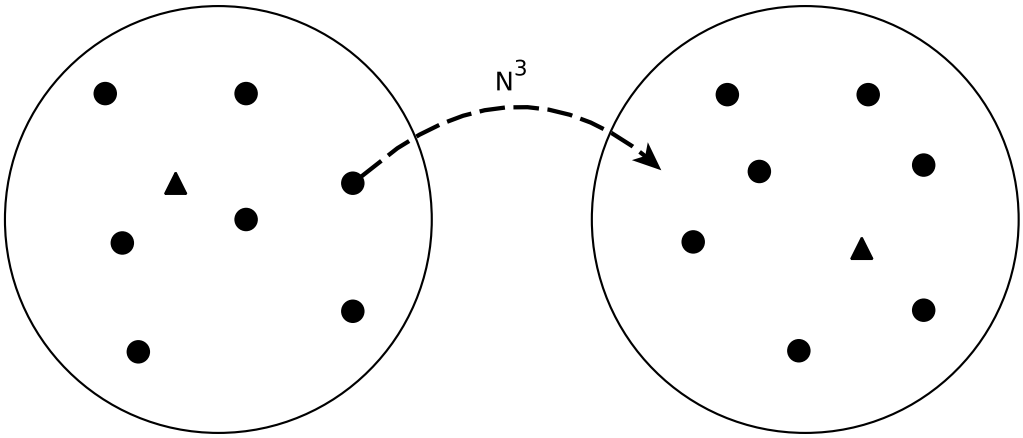

Figura 4.5: Representação de uma aplicação de um movimento de N^3.

4.2.4 Vizinhança N^4

Um movimento de N^4 é obtido ao trocar uma mediana por qualquer outro ponto.

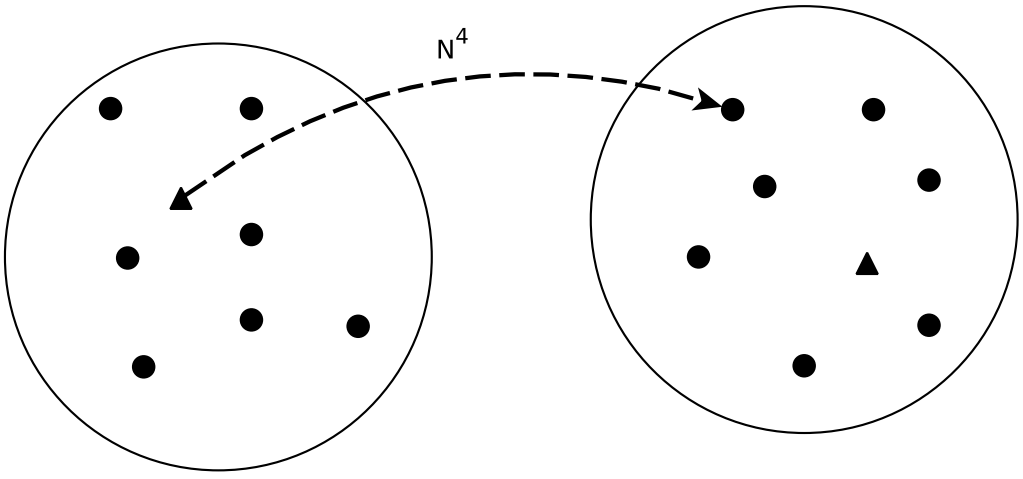

Figura 4.6: Representação de uma aplicação de um movimento de N^4.

4.2.5 Vizinhança N^5

Um movimento de N^5 é obtido ao simplesmente executar dois movimentos de N^1, em sequência.

4.2.6 Assimilação e Busca Local

Nesta seção, são descritas duas etapas da heurística de Chaves e Lorena [20]. A etapa de assimilação, que é ativada para associar a solução corrente ao *cluster* mais similar, e a etapa de busca local.

Após o laço de aplicação de movimentos, a solução resultante s é inserida no *cluster* mais similar, *i.e.*, aquele cuja solução central do *cluster* ς_i possui a menor distância para essa nova solução. Esse processo é conhecido como assimilação [78]. Sempre que uma solução é inserida em um *cluster*, um processo de reconexão por caminhos é executado entre o centro do *cluster* ς_i e s. Esse procedimento realiza o conjunto de passos necessários para ς_i se aproximar de s, *i.e.*, quais medianas devem ser trocadas para que as soluções tenham o mesmo conjunto de medianas e agrupamentos. O novo centro de *cluster* ς_i é a melhor solução encontrada nesse caminho.

Um *cluster* i é considerado promissor caso tenha atingido seu limiar de volume e, caso isso aconteça, uma busca local será ativada em ς_i. No CS-SA, a busca local é feita da seguinte maneira: para cada agrupamento, a mediana é trocada por um dos pontos a ela associado e uma realocação dos demais pontos é feita em dois momentos. No primeiro momento, os pontos que estavam alocados à mediana trocada são realocados para a mediana mais próxima. E no segundo momento, avalia-se se vale a pena realocar os pontos de outras medianas para essa nova mediana. Caso seja obtido uma melhoria no custo da solução, o procedimento é repetido em todos os outros agrupamentos até que nenhum benefício seja alcançado.

4.3 Pseudocódigo do Algoritmo CS-SA

A heurística CS-SA para o CCCP é descrita no Algoritmo 11. Inicialmente, as soluções centrais dos *clusters* são construídas, assim como a solução inicial (linhas 2 e 3). Enquanto o critério de parada do algoritmo, que corresponde a exatamente um resfriamento completo de temperatura, não é atingido, a solução corrente é explorada pelas vizinhanças N^k, $k \in [1, 5]$. No CS-SA, originalmente, um único resfriamento completo é executado e,

portanto, não há reaquecimentos. Uma vizinhança é escolhida na linha 10 e sua aplicação ocorre na linha seguinte. Caso a solução encontrada seja melhor que a solução corrente ela é aceita (linha 13). Caso contrário, a solução poderá ser aceita de acordo com a probabilidade calculada na linha 15. A temperatura é decrescida na linha 18. O *cluster* mais similar à solução corrente é calculado (linha 19), seu volume é atualizado (linha 20) e sua solução central é atualizada seguindo o processo de assimilação, que aplica uma reconexão por caminhos entre s e ς_i (linha 21). Caso o volume do cluster tenha atingido o limiar, uma busca local é aplicada (linha 24). Por fim, a melhor solução geral é atualizada (linha 26).

Algoritmo 11 CS-SA para o CCCP

1: **CS-SA** (γ, τ_{max}, T_0, T_c, ω, SA_{max})
2: Criar γ *clusters* e suas soluções centrais ς_i;
3: $s \leftarrow$ SoluçãoInicialAleatória(); $s^* \leftarrow s$;
4: **enquanto** critério de parada não satisfeito **faça**
5: $T \leftarrow T_0$;
6: **enquanto** $T > T_c$ **faça**
7: $iter \leftarrow 0$;
8: **enquanto** $iter < SA_{max}$ **faça**
9: $iter \leftarrow iter + 1$;
10: $k \leftarrow$random$[1, 5]$;
11: $s' \leftarrow N^k(s)$;
12: **se** $f(s') < f(s)$ **então**
13: $s \leftarrow s'$;
14: **senão**
15: $s \leftarrow s'$, com probabilidade $e^{\frac{-(f(s)-f(s'))}{T}}$;
16: **fim se**
17: **fim enquanto**
18: $T \leftarrow \omega T$;
19: $i \leftarrow \arg\min_{i \in \{1,\ldots,\gamma\}} \{\text{Distância}(\varsigma_i, s)\}$;
20: $\tau_i \leftarrow \tau_i + 1$;
21: $\varsigma_i \leftarrow$ Assimilacao(s, ς_i);
22: **se** $\tau_i = \tau_{max}$ **então**
23: $\tau_i = 0$;
24: $\varsigma_i \leftarrow$Busca_Local(ς_i);
25: **fim se**
26: $s^* \leftarrow min(s^*, \varsigma_i)$;
27: **fim enquanto**
28: **fim enquanto**
29: **retorne** s^*;

Na próxima seção, será apresentada a proposta de inserção da técnica de mineração de dados na heurística CS-SA, detalhando a maneira como a base de dados foi estruturada para manter as informações dos agrupamentos, e também como os padrões minerados

foram utilizados em benefício da heurística original CS-SA.

4.4 Incorporando Mineração de Dados na Heurística CS-SA

Como mencionado anteriormente, a proposta deste trabalho é incorporar mineração de dados à heurística híbrida *Clustering Search* e *Simulated Annealing* desenvolvida por Chaves e Lorena [20], que possui resultados competitivos na literatura para o CCCP, a fim de aprimorá-la.

A heurística base de Chaves e Lorena [20] pode ser dividida em duas etapas. Na primeira etapa, as soluções são geradas a partir de movimentos aplicados à solução corrente, seguindo o critério de aceitação baseado no SA. Na segunda, as soluções geradas na primeira etapa são inseridas em *clusters* para posteriormente serem exploradas, num processo conhecido como *Clustering Search*. No problema de rotulação cartográfica de pontos, descrito no Capítulo 3, a MD também foi inserida em uma heurística baseada nas metaheurísticas *Clustering Search* e *Simulated Annealing*. Assim, a construção do conjunto elite de soluções para o CCCP ocorrerá de maneira parecida à que foi empregada no trabalho anterior. As soluções serão coletadas, portanto, ao longo das iterações da heurística CS original, logo após a etapa de *Clustering Search*, representado pelas linhas 18 a 25 do Algoritmo 11. A melhor solução corrente será adicionada ao conjunto elite, por iteração, caso: (i) a solução seja melhor que a pior solução do conjunto elite, ou (ii) o conjunto elite não esteja completamente cheio (com d soluções). Em ambos os casos, somente são admitidas soluções distintas das que já estão presentes no conjunto elite.

Um dos desafios deste trabalho foi identificar qual estrutura a base de dados deveria ter de maneira que o conjunto elite pudesse ser devidamente minerado. No contexto deste problema, uma solução para o CCCP é formada por subconjuntos de elementos, em que todos os elementos sempre fazem parte da solução. O que diferencia uma solução para outra é a disposição dos elementos entre os agrupamentos. Se uma solução do CCCP fosse considerada como uma transação para o conjunto elite de soluções, todos os itens de todas as soluções sempre seriam frequentes e a extração de padrões não faria sentido.

Portanto, assim como no trabalho anterior, um mapeamento das soluções deve ser executado para gerar a base de dados de transações. Este mapeamento, que também é uma proposta do presente trabalho, é aplicado da seguinte maneira. Cada solução do CE pode ser dividida em exatamente p transações, onde p representa a quantidade

de agrupamentos. Assim, cada agrupamento de cada solução do CE se tornará uma transação da base de dados e, dessa maneira, a base de dados de transações será composta por exatamente $|CE| * p$ transações. Com isso, o minerador conseguirá identificar a ocorrência de grupos de pontos frequentes em um mesmo agrupamento, e os padrões extraídos serão caracterizados por um conjunto de pontos que ocorreram juntos em um mesmo agrupamento em pelo menos sup_{min} transações. A Figura 4.7 explica melhor a necessidade do mapeamento para este tipo de problema, ao ilustrar a estrutura de solução do CCCP em uma instância com dez pontos e três agrupamentos.

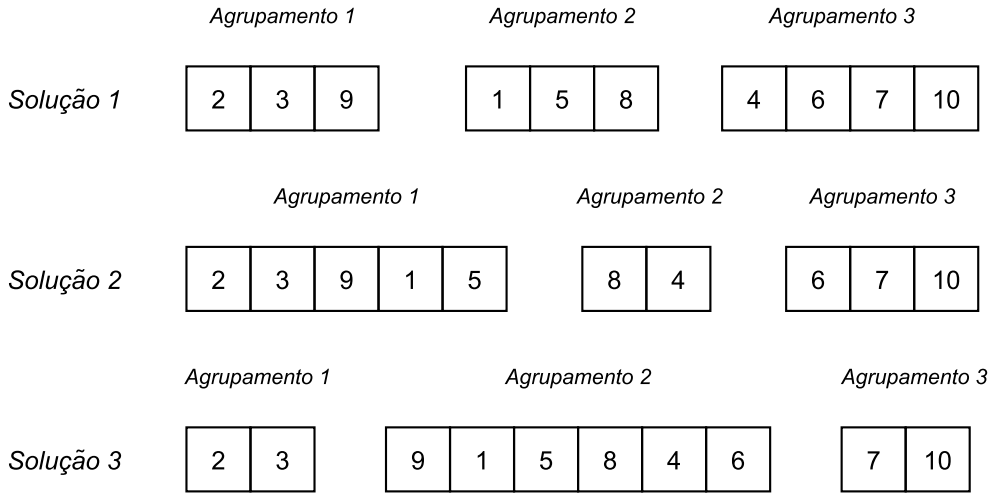

Figura 4.7: Representação de três soluções distintas do CCCP, divididas por agrupamentos.

É possível observar que cada uma das soluções é um conjunto de três subconjuntos de elementos, e o que diferencia as três soluções do CCCP são as composições dos agrupamentos. Dessa maneira, cada solução que compõe o conjunto elite é considerada, para fins da aplicação do algoritmo de mineração, como um conjunto de transações. Cada agrupamento, de cada solução elite, representará uma transação. Assim, a base de dados a ser minerada será composta por $d * p$ transações (ou agrupamentos), sendo d o número de soluções elite e p o número de agrupamentos. Um padrão minerado será representado por um conjunto de elementos que ocorreram juntos em um número mínimo de agrupamentos (transações).

Uma vez constituída a base de dados, deve-se discutir o momento da aplicação do processo de mineração e como os padrões minerados serão utilizados para guiar a busca por novas soluções. Assim, seguindo a ideia explorada na heurística DM-CS aplicada ao PRCP, optou-se por manter a aplicação do minerador sempre que o conjunto elite de

soluções se estabilizasse. Após o processo de mineração, a utilização dos padrões extraídos ocorreria no laço de aplicação das vizinhanças, dentro da componente baseada no SA. No entanto, diferentemente do trabalho anterior, a heurística CS-SA possui cinco movimentos que podem ser considerados na hora em que os padrões forem utilizados. Dentre várias possibilidades testadas para incorporação dos padrões, o movimento adaptado que mais obteve resultados interessantes foi o movimento N^6, que será detalhado na seção a seguir.

4.4.1 Vizinhança N^6

A vizinhança N^6 foi criada tendo como base a vizinhança N^3 da heurística original. Foram feitas pequenas modificações no mecanismo dos movimentos de N^6 para que o uso do padrão fosse mais efetivo. No movimento N^3 tradicional, escolhe-se um ponto aleatório que será movido para um outro agrupamento. Para o movimento de N^6, após escolher um ponto pt_{selec} aleatoriamente, busca-se o padrão de maior tamanho que contenha o ponto selecionado. Em seguida, um ponto pt_{padrao} que está contido nesse padrão é escolhido e movido para o agrupamento de pt_{selec}. Caso o ponto pt_{padrao} já esteja no mesmo agrupamento que o ponto pt_{selec}, o movimento tradicional de N^3 é aplicado. Dessa forma, o movimento de N^6 direciona os movimentos de maneira que pontos presentes em um mesmo padrão fiquem juntos também em um mesmo agrupamento dentro da solução. Assim, após a aplicação do processo de mineração, a vizinhança N^6 é incorporada à lista de vizinhanças originais da heurística CS-SA. A Figura 4.8 exemplifica a utilização do padrão.

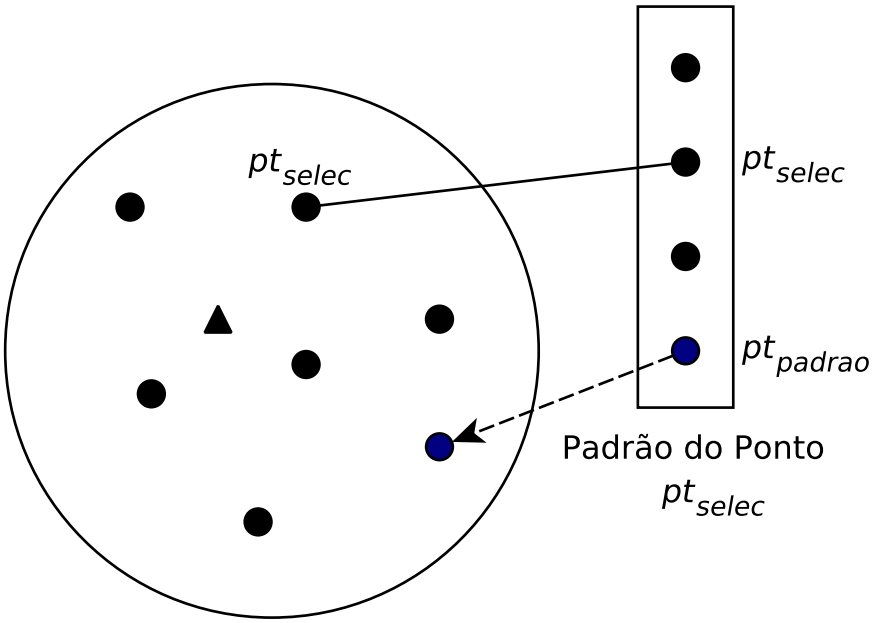

Figura 4.8: Representação de uma aplicação do movimento N^6.

4.4.2 Pseudocódigo do Algoritmo DM-Est-CS-SA

Assim como no Capítulo 3, como já mencionado anteriormente, a vizinhança N^6 começa a ser utilizada tão logo o minerador é ativado (*i.e.*, quando o CE se estabilizar) e padrões são extraídos. O CS-SA de [20] segue a estrutura clássica da metaheurística *Simulated Annealing*, realizando apenas um resfriamento. Ao final de um resfriamento, com a temperatura atual próxima da temperatura final, movimentos de piora praticamente não são mais aceitos pela heurística SA e, movimentos de melhoria ficam menos frequentes uma vez que o SA se aproxima de um ótimo local.

O Algoritmo 12 mostra a heurística híbrida com mineração de dados, que será denominada DM-Est-CS-SA. As modificações em relação ao Algoritmo 11 estão representadas nas linhas 11 a 16, 31 e 32 a 34. O conjunto elite é construído na linha 31, a mineração é executada na linha 33 e a utilização dos padrões acontece nas linhas 11 a 16. Assim como na heurística CS-SA, o critério de parada do DM-Est-CS-SA é a execução completa de um resfriamento.

Na próxima seção, são apresentados os resultados obtidos pela heurística DM-Est-CS-SA e sua comparação com a heurística original.

Algoritmo 12 Heurística Híbrida DM-Est-CS-SA para o CCCP

DM-Est-CS-SA (γ, τ_{max}, T_0, T_c, ω, SA_{max}, sup_{min}, d, u)
Criar γ *clusters* e suas soluções centrais ς_i;
$s \leftarrow$ SoluçãoInicialAleatória(); $s^* \leftarrow s$;
$CE \leftarrow \emptyset$;
enquanto critério de parada não satisfeito **faça**
  $T \leftarrow T_0$;
  enquanto $T > T_c$ **faça**
    $iter \leftarrow 0$;
    enquanto $iter < SA_{max}$ **faça**
      $iter \leftarrow iter + 1$;
      se $\neg$ ExecutouMineração() **então**
        $k \leftarrow$random$[1, 5]$;
      senão
        $k \leftarrow$random$[1, 6]$;
      fim se
      $s' \leftarrow N^k(s)$;
      se $f(s') < f(s)$ **então**
        $s \leftarrow s'$;
      senão
        $s \leftarrow s'$, com probabilidade $e^{\frac{-(f(s)-f(s'))}{T}}$;
      fim se
    fim enquanto
    $T \leftarrow \omega T$;
    $i \leftarrow \arg\min_{i \in \{1,\ldots,\gamma\}} \{\text{Distância}(\varsigma_i, s)\}$;
    $\tau_i \leftarrow \tau_i + 1$;
    $\varsigma_i \leftarrow min(s, \varsigma_i)$;
    se $\tau_i = \tau_{max}$ **então**
      $\tau_i = 0$;
      $s \leftarrow$Busca_Local(ς_i);
    fim se
    AtualizaConjuntoEliteDeSoluções(d,s,CE);
    se CE está estável **então**
      ExecutaMineração(CE, sup_{min}, u);
    fim se
    $s^* \leftarrow min(s^*, \varsigma_i)$;
  fim enquanto
fim enquanto
retorne S^*;

4.5 Resultados Computacionais

A heurística híbrida CS-SA de Chaves e Lorena [20] foi implementada na linguagem C++, bem como as heurísticas híbridas com mineração de dados propostas neste trabalho. Testes computacionais foram realizados em um computador equipado com processador Intel Core i5 CPU 650 @ 3.20GHz, com 8GB de memória RAM e Sistema Operacional Linux Ubuntu versão 14.04. Todos os experimentos foram executados em uma única *thread* e foram considerados quatro conjuntos de instâncias do CCCP utilizados em Chaves e Lorena [20]: o primeiro denominado *ta* contendo sete instâncias, o segundo denominado *SJC* contendo seis instâncias, o terceiro conjunto denominado *doni* contendo sete instâncias e, por fim, o conjunto *p3038*, com cinco instâncias. Vale ressaltar que o código original do CS-SA foi disponibilizado pelos autores, sendo utilizado como base na implementação das heurísticas híbridas com MD propostas neste trabalho.

A parametrização da heurística DM-Est-CS-SA seguiu os valores utilizados em Chaves e Lorena [20]. Os valores escolhidos para T_0, T_c, ω e SA_{max} são, respectivamente: 1000000, 0,0001, 0,95 e 1000. Além desses, os parâmetros relativos à heurística híbrida DM-Est-CS-SA proposta neste trabalho são: o tamanho do conjunto elite d, a quantidade de padrões minerados u e o valor de suporte mínimo sup_{min}. Os valores destes parâmetros foram escolhidos com base nos trabalho de Plastino *et al.* [83], sendo usados, respectivamente, 10, 100 e 8.

As heurísticas foram executadas dez vezes para cada instância da literatura, com sementes diferentes, variando de um até dez. Foram reportados os custos das melhores soluções, o custo médio das dez soluções e também o tempo médio de execução despendido. Como o código do CS-SA foi disponibilizado pelos autores, foi possível realizar testes com o código original, porém com sementes distintas das que foram empregadas em Chaves e Lorena [20], uma vez que não foi possível saber de fato quais foram as sementes usadas. Embora os resultados encontrados durante a execução do CS-SA tenham sido ligeiramente diferentes dos reportados em Chaves e Lorena [20], optou-se por apresentar apenas os resultados alcançados nestas execuções, a fim de realizar uma comparação mais justa com a heurística proposta.

Na Tabela 4.1, são apresentados os resultados computacionais obtidos pela heurística original e pela heurística híbrida com mineração de dados, tendo como critério para aplicação do minerador a estabilização do CE. Essa tabela reporta, inicialmente, o melhor valor conhecido na literatura para cada uma das instâncias (coluna BKS). Em seguida,

para cada algoritmo, a melhor solução obtida, o valor médio de solução e o tempo computacional médio das heurísticas CS-SA e DM-Est-CS-SA com Estabilização, relativos às dez execuções, são reportados. Na comparação entre os algoritmos, os valores em negrito representam os melhores resultados obtidos em cada critério.

Tabela 4.1: Resultados computacionais do CS-SA e DM-Est-CS-SA

				CS-SA			DM-Est-CS-SA com Estabilização				
Instância	n	p	BKS	MelhorSol	MédiaSol	Tempo	MelhorSol	Δ %	MédiaSol	Δ %	Tempo
ta25	25	5	1251.45$^{\upsilon}$	1251.45	1251.45	0.29	1251.45	0.00	1251.45	0.00	0.21
ta50	50	5	4474.52$^{\delta}$	4474.52	4474.52	0.56	4474.52	0.00	4474.52	0.00	0.37
ta60	60	5	5356.58$^{\upsilon}$	5356.58	5356.58	0.66	5356.58	0.00	5356.58	0.00	0.44
ta70	70	5	6240.67$^{\delta}$	6240.67	6240.67	0.75	6240.67	0.00	6240.67	0.00	0.47
ta80	80	7	5515.46$^{\delta}$	5730.28	5730.28	1.16	5730.28	0.00	5730.28	0.00	0.69
ta90	90	4	8899.05$^{\delta}$	9069.85	9069.85	0.96	9069.85	0.00	9069.85	0.00	0.56
ta100	100	6	8102.04$^{\zeta}$	8102.04	8103.88	1.41	8102.04	0.00	**8103.41**	-0.01	0.80
SJC1	100	10	17359.75$^{\zeta}$	17363.47	17363.47	1.85	17363.47	0.00	17363.47	0.00	1.12
SJC2	200	15	33181.65$^{\zeta}$	33181.65	33181.96	4.45	33181.65	0.00	33181.96	0.00	4.31
SJC3a	300	25	45358.23$^{\theta}$	45372.09	**45440.22**	14.95	45372.09	0.00	45447.00	0.01	16.59
SJC3b	300	30	40661.94$^{\theta}$	40706.80	**40759.40**	18.44	**40695.47**	-0.03	40762.33	0.01	20.44
SJC4a	402	30	61931.60$^{\theta}$	61969.76	**62032.56**	31.91	**61949.96**	-0.03	62038.95	0.01	34.06
SJC4b	402	40	52214.55$^{\zeta}$	52301.15	52373.17	265.46	52301.15	0.00	52373.17	0.00	251.39
doni1	1000	6	3021.41$^{\upsilon}$	3087.36	3110.35	8.97	3087.36	0.00	3110.36	0.00	14.5
doni2	2000	6	6080.70$^{\upsilon}$	6431.62	**6452.10**	18.98	6431.62	0.00	6460.09	0.12	28.12
doni3	3000	8	8438.96$^{\theta}$	8670.36	8881.50	35.47	8670.36	0.00	**8879.26**	-0.03	45.82
doni4	4000	10	10854.48$^{\zeta}$	**11164.41**	**11308.62**	119.35	11255.48	0.81	11344.45	0.32	126.44
doni5	5000	12	11134.94$^{\zeta}$	11188.86	**11256.27**	209.97	11188.86	0.00	11265.35	0.08	217.27
doni6	10000	23	15722.67$^{\theta}$	15815.43	15976.41	1938.11	15815.43	0.00	15976.41	0.00	1913.26
doni7	13221	30	18596.74$^{\theta}$	18986.20	19289.18	5304.69	18986.20	0.00	19289.18	0.00	5285.96
p3038-600	3038	600	128419.95$^{\theta}$	128927.85	**130050.18**	2030.34	128927.85	0.00	130132.36	0.06	2048.12
p3038-700	3038	700	116325.05$^{\theta}$	**116869.45**	**117665.63**	2367.21	117069.79	0.17	117777.01	0.09	2364.30
p3038-800	3038	800	107764.69$^{\theta}$	108452.04	109237.44	2714.82	**107756.85**	-0.65	**108978.25**	-0.24	2710.70
p3038-900	3038	900	99968.15$^{\theta}$	**100711.58**	**101368.23**	3108.82	100819.65	0.11	101411.69	0.04	3079.29
p3038-1000	3038	1000	92706.38$^{\theta}$	**93751.23**	**94814.83**	3480.40	94115.73	0.39	94845.27	0.03	3450.36

$^{\upsilon}$: Negreiros e Palhano (2006) $^{\delta}$: Pereira e Senne (2008)

$^{\zeta}$: Chaves e Lorena (2010) $^{\theta}$: Chaves e Lorena (2011)

Observando a Tabela 4.1, é possível perceber que, para o grupo de instâncias pequenas (*ta*), ambas as heurísticas chegaram exatamente nas mesmas melhores soluções, com uma vantagem para o DM-Est-CS-SA em relação à media de custo de solução. Em relação ao grupo de instâncias *SJC*, é possível verificar que, em relação à melhor solução, o desempenho do DM-Est-CS-SA com Estabilização foi superior ao CS-SA em duas soluções. Por outro lado, o CS-SA obteve três melhores médias. Para o penúltimo grupo, a heurística CS-SA obteve uma melhor solução e três melhores médias, enquanto que a heurística DM-Est-CS-SA obteve apenas uma melhor média de solução. Para o último grupo de

instâncias, a heurística original obteve os melhores resultados, alcançando três melhores soluções e quatro melhores médias de solução, contra apenas uma melhor solução e uma melhor média da heurística proposta.

Os resultados obtidos pela heurística DM-Est-CS-SA, realizando a mineração assim que o CE estivesse estável, foram pouco expressivos. Isso impulsionou o estudo para encontrar um melhor momento para aplicar o minerador. Na próxima seção, uma nova heurística híbrida com mineração de dados será introduzida, modificando a versão DM-Est-CS-SA.

4.6 Heurística DM-CS-SA

A heurística DM-Est-CS-SA, proposta na seção anterior, não obteve resultados expressivos em relação à heurística base CS-SA. Diferentemente da abordagem detalhada no Capítulo 3, em que a heurística CS executava vários resfriamentos, a abordagem CS-SA deste capítulo executa apenas um resfriamento. Assim, como a estabilização do CE acontecia sempre ao final do resfriamento, com a temperatura baixa, poucas iterações restavam para o uso do padrão. No Capítulo 3, as Figuras 3.5 e 3.6 também deram indícios de que a aplicação dos padrões em um único resfriamento (no caso, o primeiro resfriamento) não resultaria em benefícios para o SA. Naquela análise, somente após o segundo resfriamento (e consequentemente a segunda aplicação do minerador) que foi possível perceber que a aplicação dos padrões beneficiou a heurística híbrida proposta. Portanto, acredita-se que a mineração seja mais proveitosa se realizada ao final de um resfriamento, minerando um conjunto elite que possua soluções melhores (possivelmente ótimos locais), em vez de realizar o processo de extração de padrões ao longo do primeiro resfriamento, onde a heurística ainda não teria alcançado soluções promissoras. Logo, esta seção introduz uma nova proposta de hibridização da heurística CS-SA com mineração de dados, denominada DM-CS-SA, explorando dois resfriamentos de temperatura da heurística original.

Abramson *et al.* [3] discutiram várias estratégias de resfriamento e reaquecimentos no valor de temperatura, e como essas estratégias poderiam ser aplicadas à metaheurística *Simulated Annealing*. Seguindo uma variação das alternativas propostas em [3], optou-se por realizar, para a nova proposta de heurística híbrida com MD, um reaquecimento completo da temperatura (*i.e.*, $T \leftarrow T_0$), utilizando como solução inicial desta nova etapa de resfriamento a última solução alcançada no resfriamento anterior. Nesse cenário, a aplicação do processo de mineração ocorrerá entre os dois resfriamentos, usando o primeiro

resfriamento para o preenchimento do conjunto elite de soluções e o segundo para a aplicação dos padrões extraídos. Assim, nessa nova abordagem, o primeiro resfriamento seguirá aplicando os movimentos das vizinhanças N^1 a N^5 e, para o segundo, inclui-se a nova vizinhança N^6, que faz uso dos padrões.

O Algoritmo 13 apresenta o pseudocódigo da heurística DM-CS-SA. A principal diferença para o Algoritmo 12 é que a mineração é agora aplicada na linha 34, após o primeiro resfriamento, e não depende da estabilização do CE. O critério de parada do DM-CS-SA é a execução completa de dois resfriamentos. Vale ressaltar que, embora tenha dois resfriamentos, a heurística DM-CS-SA executa exatamente a mesma quantidade de iterações que as heurísticas DM-Est-CS-SA e CS-SA. A quantidade de iterações dessas heurísticas é definida pela multiplicação da temperatura inicial T_0 pela taxa de resfriamento ω. Assim, para que a heurística DM-CS-SA tivesse a mesma quantidade de resfriamentos que as outras heurísticas, sua taxa de resfriamento foi alterada para ω^2 (linha 23 do Algoritmo 13), garantindo que cada resfriamento de DM-CS-SA realizasse metade das iterações das outras heurísticas.

Na Tabela 4.2, que possui a mesma estrutura da Tabela 4.1, estão os resultados computacionais obtidos pela heurística original CS-SA e pela heurística híbrida com mineração de dados com dois resfriamentos DM-CS-SA. O ambiente de execução dessas heurísticas é o mesmo que foi descrito na Seção 4.5. Na comparação entre os algoritmos, os valores em negrito representam os melhores resultados obtidos em cada critério.

Algoritmo 13 Heurística Híbrida DM-CS-SA para o CCCP

DM-CS-SA (γ, τ_{max}, T_0, T_c, ω, SA_{max}, sup_{min}, d, u)
Criar γ *clusters* e suas soluções centrais ς_i;
$s \leftarrow$ SoluçãoInicialAleatória(); $s^* \leftarrow s$;
$CE \leftarrow \emptyset$;
enquanto critério de parada não satisfeito **faça**
  $T \leftarrow T_0$;
  enquanto $T > T_c$ **faça**
    $iter \leftarrow 0$;
    enquanto $iter < SA_{max}$ **faça**
      $iter \leftarrow iter + 1$;
      se $\neg$ ExecutouMineração() **então**
        $k \leftarrow$random$[1, 5]$;
      senão
        $k \leftarrow$random$[1, 6]$;
      fim se
      $s' \leftarrow N^k(s)$;
      se $f(s') < f(s)$ **então**
        $s \leftarrow s'$;
      senão
        $s \leftarrow s'$, com probabilidade $e^{\frac{-(f(s)-f(s'))}{T}}$;
      fim se
    fim enquanto
    $T \leftarrow \omega^2 T$;
    $i \leftarrow \underset{i \in \{1,\ldots,\gamma\}}{\arg\min}$ {Distância(ς_i, s)};
    $\tau_i \leftarrow \tau_i + 1$;
    $\varsigma_i \leftarrow min(s, \varsigma_i)$;
    se $\tau_i = \tau_{max}$ **então**
      $\tau_i = 0$;
      $s \leftarrow$Busca_Local(ς_i);
    fim se
    AtualizaConjuntoEliteDeSoluções(d,s,CE);
    $s^* \leftarrow min(s^*, \varsigma_i)$;
  fim enquanto
  ExecutaMineração(CE, sup_{min}, u);
fim enquanto
retorne S^*;

Tabela 4.2: Resultados computacionais do CS-SA e DM-CS-SA

				CS-SA			DM-CS-SA				
Instância	n	p	BKS	MelhorSol	MédiaSol	Tempo	MelhorSol	Δ %	MédiaSol	Δ %	Tempo
ta25	25	5	1251.45	1251.45	1251.45	0.29	1251.45	0.00	1251.45	0.00	0.46
ta50	50	5	4474.52	4474.52	4474.52	0.56	4474.52	0.00	4474.52	0.00	0.74
ta60	60	5	5356.58	5356.58	5356.58	0.66	5356.58	0.00	5356.58	0.00	0.86
ta70	70	5	6240.67	6240.67	6240.67	0.75	6240.67	0.00	6240.67	0.00	0.98
ta80	80	7	5515.46	5730.28	5730.28	1.16	5730.28	0.00	5730.28	0.00	1.42
ta90	90	4	8899.05	9069.85	9069.85	0.96	9069.85	0.00	9069.85	0.00	1.18
ta100	100	6	8102.04	8102.04	8103.88	1.41	8102.04	0.00	**8102.52**	-0.02	1.72
SJC1	100	10	17359.75	17363.47	**17363.47**	4.85	**17359.75**	-0.02	17364.02	0.00	6.99
SJC2	200	15	33181.65	33181.65	33181.96	19.45	33181.65	0.00	**33181.65**	0.00	21.61
SJC3a	300	25	45358.23	45372.09	45440.22	74.95	**45354.28**	-0.04	**45439.45**	0.00	79.68
SJC3b	300	30	40661.94	40706.80	40759.40	100.44	**40690.57**	-0.04	**40758.11**	0.00	104.51
SJC4a	402	30	61931.60	61969.76	62032.56	177.91	**61937.94**	-0.05	**62028.83**	-0.01	188.62
SJC4b	402	40	52214.55	52301.15	52373.17	265.46	**52300.23**	0.00	**52358.29**	-0.03	280.97
doni1	1000	6	3021.41	3087.37	3110.36	8.97	**3079.29**	-0.26	**3100.06**	-0.33	10.73
doni2	2000	6	6080.70	**6431.62**	**6452.10**	18.98	6447.95	0.25	6477.87	0.40	22.55
doni3	3000	8	8438.96	**8670.36**	**8881.50**	35.47	8805.46	1.53	8903.43	0.25	42.43
doni4	4000	10	10854.48	11164.41	11308.62	119.35	**11151.86**	-0.11	**11266.57**	-0.37	127.30
doni5	5000	12	11134.94	**11188.86**	**11256.27**	209.97	11223.43	0.31	11288.23	0.28	216.17
doni6	10000	23	15722.67	**15815.43**	15976.41	1938.11	15879.61	0.40	**15971.59**	-0.03	1890.28
doni7	13221	30	18596.74	18986.20	19289.18	5304.69	**18754.30**	-1.24	**19244.78**	-0.23	5403.88
p3038-600	3038	600	128419.95	**128927.85**	**130050.18**	2030.34	130666.84	1.33	131128.92	0.82	2027.53
p3038-700	3038	700	116325.05	**116869.43**	**117665.63**	2367.21	117853.13	0.83	118291.18	0.53	2381.97
p3038-800	3038	800	107764.69	108452.04	109237.44	2714.82	**107987.42**	-0.43	**109211.13**	-0.02	2778.37
p3038-900	3038	900	99968.15	100711.58	**101368.23**	3108.82	**100624.79**	-0.09	101487.93	0.12	3092.49
p3038-1000	3038	1000	92706.38	**93751.23**	94814.83	3480.40	94112.64	0.38	**94786.08**	-0.03	3457.17

Observando a Tabela 4.2, é possível perceber que, para o grupo de instâncias pequenas (*ta*), ambas as heurísticas chegaram exatamente nas mesmas melhores soluções, com o DM-CS-SA obtendo apenas uma melhor média de solução. Em relação ao grupo de instâncias *SJC*, é possível verificar que o desempenho do DM-CS-SA foi superior ao CS-SA, uma vez que encontrou cinco melhores médias de solução em um total de seis instâncias, além de conseguir reportar cinco melhores soluções, contra um empate. Vale

ressaltar que, para a instância SJC3a, o DM-CS-SA reportou a melhor solução da literatura, considerando todas as abordagens para o CCCP. Para o conjunto de instâncias *doni*, o DM-CS-SA conseguiu melhores médias de solução em quatro de sete instâncias, perdendo em três. Nas melhores soluções ganhou em três e perdeu em quatro instâncias. No conjunto final de instâncias *p3038*, ganhou em duas e perdeu em três para a melhor solução e para a média de solução.

Nos quatro grupos de instâncias maiores, o algoritmo proposto conseguiu dez melhores soluções contra sete da estratégia em comparação. Com relação à média de custo de solução, a heurística híbrida com mineração conseguiu doze melhores médias contra sete da heurística original, em um total de 25 instâncias. Tais resultados foram obtidos em um tempo de execução muito próximo em relação à heurística base original. Embora os resultados indiquem que o benefício da introdução de MD na heurística DM-CS-SA também não tenha sido muito relevante, principalmente se comparados aos resultados obtidos pela proposta híbrida com MD do Capítulo 3, foi possível perceber uma leve evolução de desempenho entre as heurísticas DM-Est-CS-SA e DM-CS-SA.

Outro fator que pode ter influenciado os resultados das heurísticas DM-Est-CS-SA e DM-CS-SA é o fato de que o CS-SA utiliza duas maneiras distintas de avaliar as soluções. A primeira maneira realiza o cálculo real da função objetivo, enquanto que a segunda maneira realiza um cálculo aproximado com uma função objetivo guia, tomando as medianas como base para esse cálculo. Assim, as soluções coletadas para o CE são avaliadas com a função objetivo real, logo após a etapa de *Clustering*. Porém, no momento em que os padrões são utilizados (i.e., nos movimentos da vizinhança N^6), as soluções geradas são avaliadas pela função objetivo aproximada. Isso pode ter afetado negativamente a utilização dos padrões, uma vez que as duas funções nem sempre fornecem o mesmo resultado nas comparações de soluções.

Para avaliar o comportamento da heurística DM-CS-SA, um experimento foi realizado reportando o custo de solução encontrado por iteração dessa heurística. Para esse experimento, optou-se por comparar o DM-CS-SA com uma versão alternativa da heurística CS-SA, que realiza dois resfriamentos, denominada 2R-CS-SA. Dessa maneira, é possível observar o comportamento do custo de solução das duas heurísticas ao longo do segundo resfriamento, buscando-se avaliar o benefício da introdução da vizinhança N^6 que utiliza os padrões minerados. Nesse experimento foi realizado uma única execução das heurísticas 2R-CS-SA e DM-CS-SA para as instâncias `ta60`, `sjc3a`, `doni2` e `doni5`, e os resultados são apresentados nas Figuras 4.9 a 4.12. É possível perceber que o comportamento das

duas heurísticas 2R-CS-SA e DM-CS-SA é exatamente igual no primeiro resfriamento, como já era esperado. A partir do segundo resfriamento, quando os padrões minerados começam a ser utilizados pelo movimentos adaptado N^6, fica evidente a melhoria no custo de solução em todas as figuras.

No próximo capítulo, será apresentado o Problema de Dispersão de Arquivos em Serviços de Armazenamentos em Nuvem. Para resolver esse problema, são propostas uma formulação matemática, baseada em programação linear inteira, e duas heurísticas baseadas nas metaheurísticas SA e GRASP.

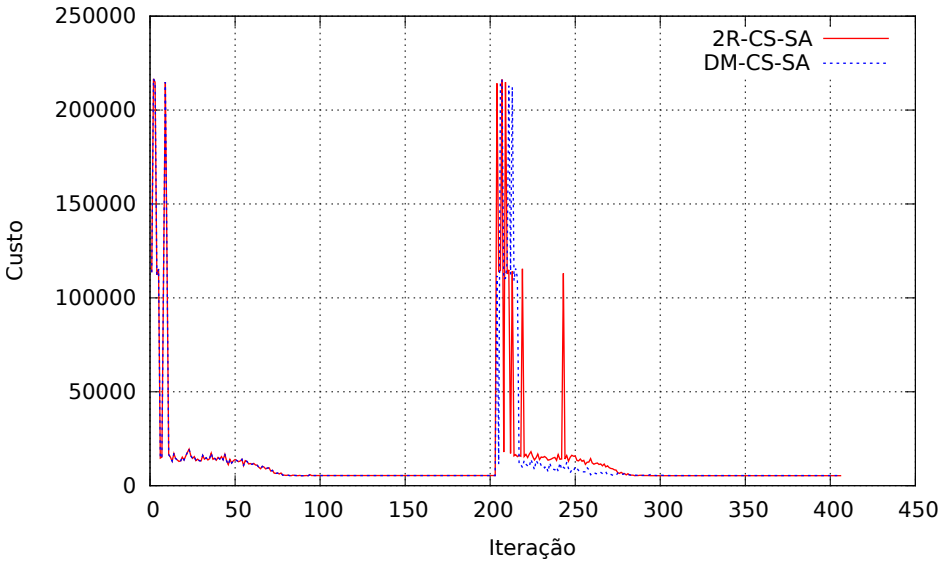

Figura 4.9: Gráficos de Custo x Iteração - Instância ta60.

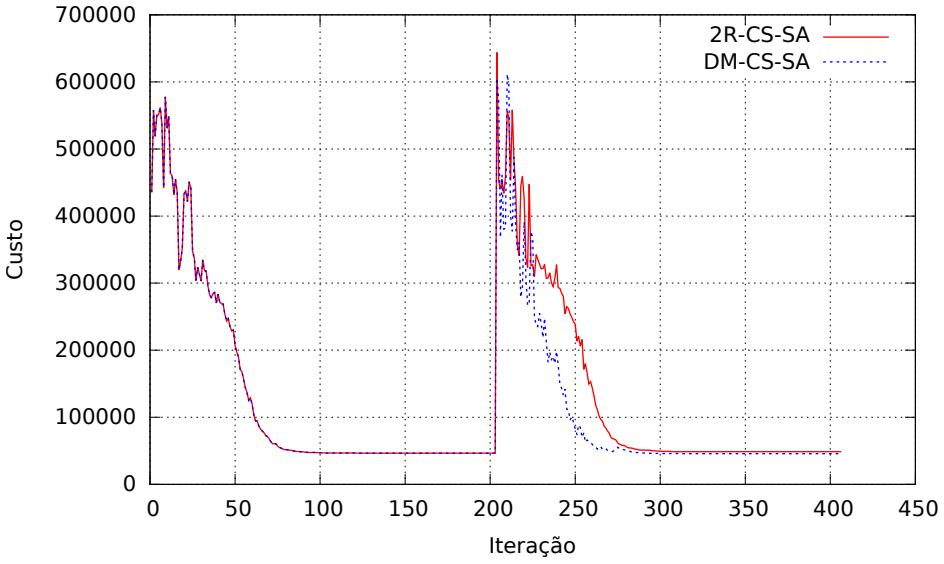

Figura 4.10: Gráficos de Custo x Iteração - Instância SJC3a.

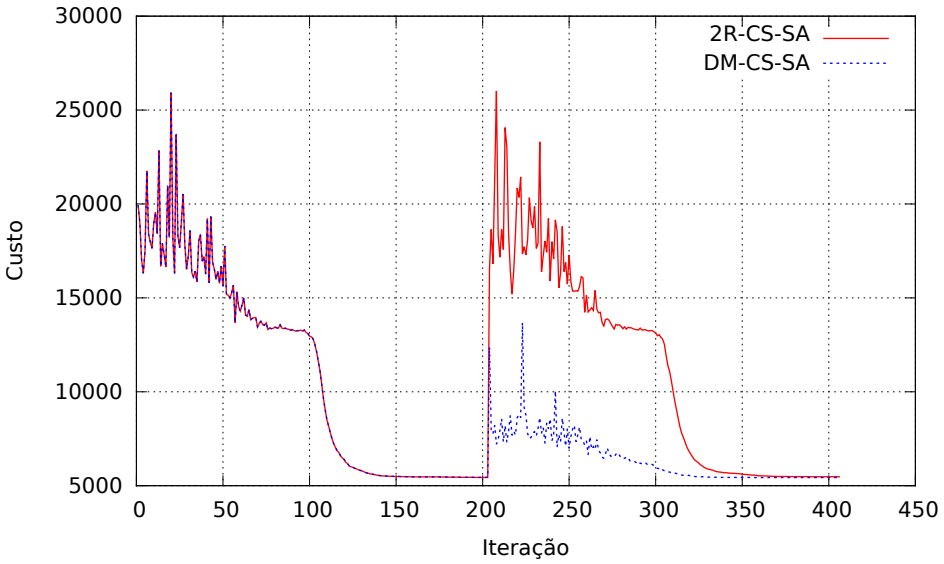

Figura 4.11: Gráficos de Custo x Iteração - Instância doni2.

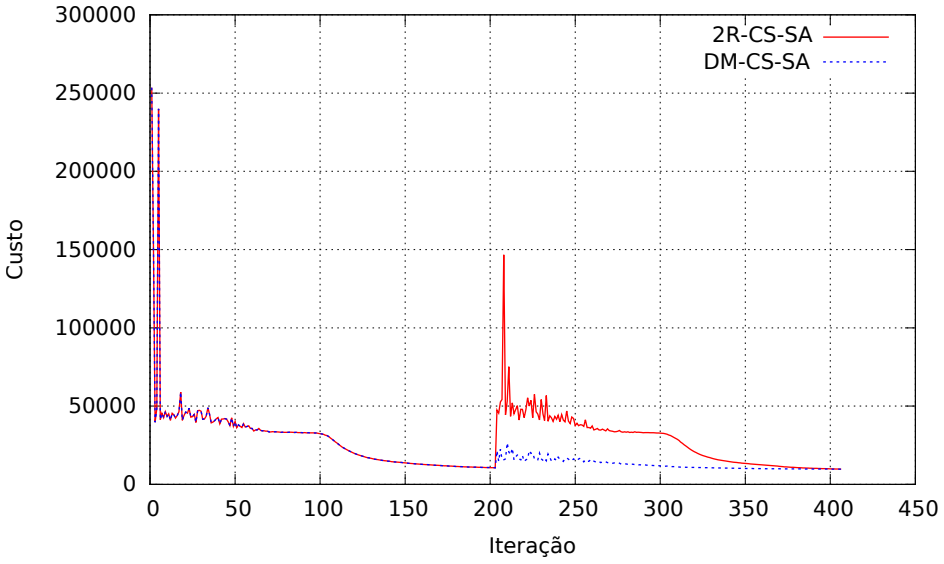

Figura 4.12: Gráficos de Custo x Iteração - Instância doni5.

Capítulo 5

Problema de Dispersão de Arquivos em Serviços de Armazenamento em Nuvem

5.1 Introdução

Nos últimos anos, *workflows* científicos (ou somente *workflows*) têm sido largamente utilizados como uma abstração que modela os passos de uma simulação científica [69, 104]. Tais passos envolvem a execução de modelos complexos com diversos parâmetros distintos. *Workflows* são geralmente modelados como grafos de atividades, de maneira que cada nó do grafo representa um processamento de dados e cada arco representa dependências de dados entre atividades [104]. Cada atividade em um *workflow* está associada a um programa, *script* ou serviço web [69]. A execução de uma atividade específica com um conjunto de valores para cada parâmetro é conhecida como ativação [77].

Workflows podem ser implementados de diversas maneiras, tais como scripts em *bash* ou em Python [73]. Entretanto, eles são comumente modelados e executados em ambientes conhecidos como *Scientific Workflow Management Systems* (SWfMS) [104]. SWfMS suportam composição, execução, monitoramento e captura de proveniência dos *workflows* [35]. Muitas áreas do conhecimento usam *workflows* científicos e ambientes do tipo SWfMS em suas atividades diárias, como, por exemplo, biologia [76], química [100] e astronomia [10, 29]. Diversos *workflows* existentes produzem e consomem grandes quantidades de dados e, ainda, são executados várias vezes até que o cientista responsável possa confirmar ou refutar sua hipótese no experimento. Devido ao grande volume de dados envolvidos nessas execuções, além da alta demanda de processamento, esses tipos de *workflows* precisam de ambientes de computação com alta performance (*High Performance Computing* (HPC)) e/ou ambientes com alta capacidade de escalabilidade (*Data-*

Intensive Scalable Computing) [16] fornecendo, também, possibilidade de paralelismo para que se possa produzir resultados em um tempo computacional viável. Ambientes com tais propriedades geralmente são obtidos com supercomputadores ou *clusters* [49, 50].

Embora *clusters* e supercomputadores sejam muito utilizados pela comunidade científica atualmente, muitos cientistas têm migrado seus *workflows* para serviços na nuvem [25, 105]. Serviços na nuvem oferecem diversos recursos com o foco em HPC e que podem ser usados em *workflows* que fazem alto consumo de dados e de processamento [53]. Serviços na nuvem também oferecem máquinas virtuais e armazenamento. Diversos SWfMS já fornecem a possibilidade de execução de *workflows*, tais como Pegasus [27], Swift/T [108], eScience Central [107], Triana [103], Galaxy [4], OpenAlea [84], e SciCumulus [26].

Quando um *workflow* é executado na nuvem, os ambientes de gerenciamento SWfMS normalmente dependem dos seus serviços de armazenamento para guardar os dados de entrada, dados intermediários e também os resultados dos experimentos. Esses dados são armazenados em *buckets* (regiões ou espaços de alocação disponibilizados pelo serviço). Por exemplo, se um SWfMS específico está sendo executado no *Amazon AWS Cloud*, os arquivos gerados por essa execução são usualmente armazenados no serviço *Amazon S3*[1]. De fato, existem SWfMS que já fornecem essas capacidades, *e.g.,* Pegasus já provê um cliente[2] para armazenar dados no serviço *Amazon S3*. De maneira similar, SciCumulus também faz uso do s3fs[3] para armazenar e recuperar dados considerando o *Amazon S3*.

Existem duas vantagens de se utilizar serviços de armazenamento em nuvem para armazenar dados que forem consumidos ou produzidos por *workflows*: (i) como o *workflow* já está sendo executado em nuvem, além de se evitar gargalos devido à transferências de arquivos, também se economiza com os custos envolvidos e (ii) esses serviços apresentam um bom custo-benefício para os cientistas, uma vez que não há a necessidade de se contratar serviços de alto custo de armazenamento, pois a responsabilidade de armazenamento e gestão dos dados é delegada para o provedor em nuvem.

Ainda que esses serviços facilitem a gestão dos arquivos gerados por *workflows* executados em nuvem, não há praticamente nenhuma garantia de confidencialidade desses arquivos e, consequentemente, dos resultados do *workflows* [28, 45, 65]. De fato, muitos cientistas evitam migrar seus experimentos para nuvem devido a esses problemas de confidencialidade. Branco Jr. *et al.* [15] discutem três tipos de abordagens que podem

[1]https://aws.amazon.com/pt/s3/details/
[2]https://pegasus.isi.edu/documentation/cli-pegasus-s3.php
[3]https://github.com/s3fs-fuse/s3fs-fuse

ser implementadas para garantir a confidencialidade dos dados armazenados em nuvem: (i) criptografia, (ii) fragmentação dos dados e (iii) uma combinação dos dois anteriores. No contexto deste trabalho, criptografia dos dados envolve, obviamente, criptografar todos os dados consumidos e produzidos pelo *workflow*. Essa criptografia pode acontecer tanto pelo lado do cliente, quanto do lado do servidor. Pelo lado do servidor, o próprio serviço de armazenamento em nuvem já criptografa os arquivos antes de salvá-los (S3, por exemplo, fornece essa possibilidade[4]). No lado do cliente, o próprio usuário (ou o ambiente SWfMS) deve realizar a criptografia dos dados antes de enviar para a nuvem. Embora a criptografia seja relativamente simples de implementar, gerenciar diversas chaves criptográficas para diferentes serviços de armazenamento em nuvem pode se tornar um outro problema, de acordo com Saramati *et al.* [97]. Fragmentação de dados envolve dividir dados sensíveis em diversos fragmentos, que podem ser armazenados em diferentes serviços de armazenamento em nuvem. Apesar de ser uma ideia promissora, o ato de fragmentar dados científicos pode se tornar uma tarefa complexa uma vez que os dados são geralmente heterogêneos e envolvem complexas estruturas de armazenamento, tais como *Quadtrees* ou *Octrees*. Sousa *et al.* [102] discutem sobre a complexidade de se identificar regiões de interesse em dados científicos heterogêneos para a fragmentação.

Não obstante, todas as abordagens anteriores estão desconectadas da abstração do *workflow* e também do SWfMS, *i.e.*, elas não consideram a associação e a semântica dos arquivos produzidos pelas atividades do *workflow*, o que pode ser um problema. Para ilustrar esse problema, considera-se o *workflow* SciPhy [76] como um exemplo (Figura 5.1). SciPhy é um *workflow* para análise filogenética que busca produzir árvores filogenéticas tendo como entrada dados de DNA, RNA e sequência de aminoácidos. SciPhy é composto de quatro atividades: (i) alinhamento de sequência (implementado pelo programa MAFFT), (ii) conversão de alinhamento (realizada pelo programa ReadSeq), (iii) escolha do modelo evolucionário (executado pelo programa ModelGenerator) e (iv) geração da árvore (implementado pelo programa RAxML). Para cada sequência de entrada de DNA ou RNA, a abordagem MAFFT produz um arquivo "*.mafft", o programa ReadSeq produz um arquivo "*.phylip", ModelGenerator produz um arquivo "*.mg.modelFromMG.txt" e RAxML produz os arquivos "RAxML_result.x.phylip_raxml_tree1.singleTree" e "RAxML _bootstrap.x.phylip_tree2.raxml". Esses arquivos podem conter, por exemplo, dados sensíveis (às vezes ainda não publicados) de doenças de um grupo de pacientes. Logo, seguindo protocolos bioéticos, se faz necessário manter a privacidade do conteúdo desses arquivos. No *workflow* SciPhy, se esses arquivos estiverem armazenados em um mesmo

[4]docs.aws.amazon.com/AmazonS3/latest/dev/UsingEncryption.html

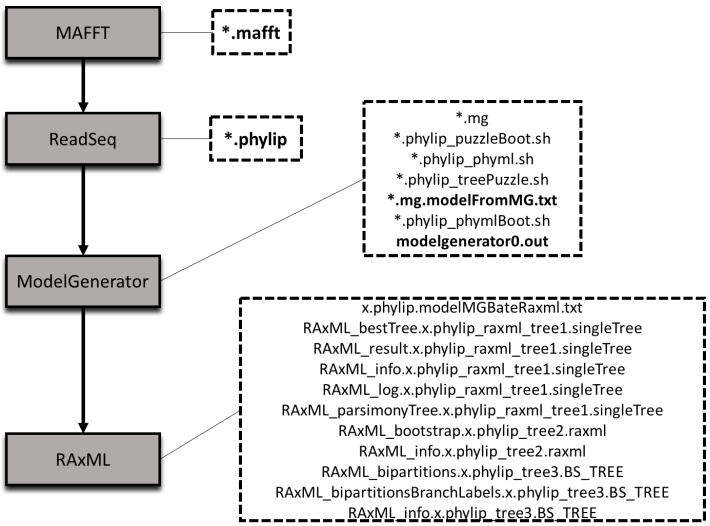

Figura 5.1: *Workflow* SciPhy e os arquivos gerados por cada atividade.

bucket, a segurança da pesquisa pode ser comprometida. Se uma pessoa não autorizada obtiver acesso a esses dados, terá total acesso aos experimentos (e também à estrutura do *workflow*). Mesmo se esses arquivos fossem distribuídos arbitrariamente entre *buckets*, alguns arquivos ainda poderiam ser armazenados juntos, o que continuaria não resolvendo o problema de confidencialidade dos resultados. Portanto, alguns arquivos específicos produzidos e consumidos devem ser armazenados em *buckets* (ou até mesmo serviços) diferentes, de acordo com a semântica do *workflow*, *i.e.*, os especialistas devem informar quais arquivos não poderão ser armazenados juntos para que se aumente a confidencialidade dos resultados.

Nesse contexto, encontrar um plano de distribuição para dispersar os arquivos entre vários serviços de armazenamento em nuvem, levando-se em consideração todas as restrições envolvidas, tais como os possíveis conflitos entre os arquivos, os tamanhos dos arquivos e as capacidades dos *buckets*, é de fato uma tarefa difícil e pode ser modelado como um problema de otimização combinatória. Neste capítulo, inicialmente serão discutidas as possíveis especificações, restrições e outros detalhes, de maneira a fornecer uma descrição completa do problema, propondo uma definição formal como um problema de otimização combinatória. Em seguida, uma formulação matemática é proposta utilizando programação linear inteira. Além disso, são propostas duas heurísticas, uma baseada na metaheurística GRASP e uma baseada na metaheurística *Simulated Annealing*, para resolver o problema. Outra contribuição deste trabalho foi produzir o conjunto de instâncias, que foi integralmente gerado a partir de vários *workflows* públicos existentes, todos eles utilizados em aplicações reais [13][5].

[5]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

O restante deste capítulo está organizado da seguinte maneira. A Seção 5.2 apresenta a definição do problema, suas restrições e, finalmente, a formulação matemática proposta. A Seção 5.3 apresenta o ambiente de testes, as instâncias e exibe os resultados computacionais conduzidos com o modelo. Finalmente, na Seção 5.4, as heurísticas propostas são descritas e seus resultados computacionais são comparados.

5.2 Definição do Problema

Problemas de otimização combinatória são concebidos a partir de contextos diversos, quase sempre motivados por aplicações reais. Tais problemas são comumente difíceis de resolver, se considerado seus espaços de busca exponenciais, especialmente quando dados reais estão sendo analisados e processados [23]. O contexto deste trabalho foi motivado por uma demanda proveniente de um grupo de pesquisa, liderado pelo Professor Daniel de Oliveira (IC-UFF), que possui diversos experimentos modelados com *workflows* científicos, alguns desses sendo executados em nuvem. Em seus experimentos, prover confidencialidade aos seus dados e, principalmente, aos resultados das execuções dos *workflows* sempre foi um problema. Portanto, o objetivo deste trabalho é desenvolver uma abordagem que possa garantir a confidencialidade dos dados ou, pelo menos, minimizar o risco de ter informações sensíveis acessadas por terceiros não autorizados. Assim, percebeu-se que criar um plano de distribuição de arquivos em serviços de armazenamento em nuvem, que considere características reais do problema, poderia ser uma boa alternativa para minimizar o risco uma vez que seria viável do ponto de vista prático.

Para que se obtenha um plano de distribuição dos arquivos em serviços de armazenamento em nuvem, é necessário que os possíveis conflitos entre arquivos sejam fornecidos pelo especialista *a priori*. Essas restrições podem ser apresentadas como um grafo de conflitos, em que os nós representam os arquivos consumidos/produzidos por um *workflow* e as arestas representam a ligação entre arquivos que não devem estar armazenados em um mesmo *bucket* (ou em um mesmo serviço). A Figura 5.2 apresenta como exemplo um possível grafo de conflitos. Os valores nas arestas podem ser interpretadas como penalidades, que serão computadas caso os arquivos que ligam essas arestas sejam configurados para serem armazenados juntos. Percebe-se que, dessa maneira, a restrição entre os arquivos pode ser "relaxada", o que resulta em conflitos violáveis (mediante penalidade) e conflitos invioláveis (arquivos nunca poderão ser armazenados juntos).

Entretanto, nem sempre é possível definir o valor dessas penalidades para conflitos

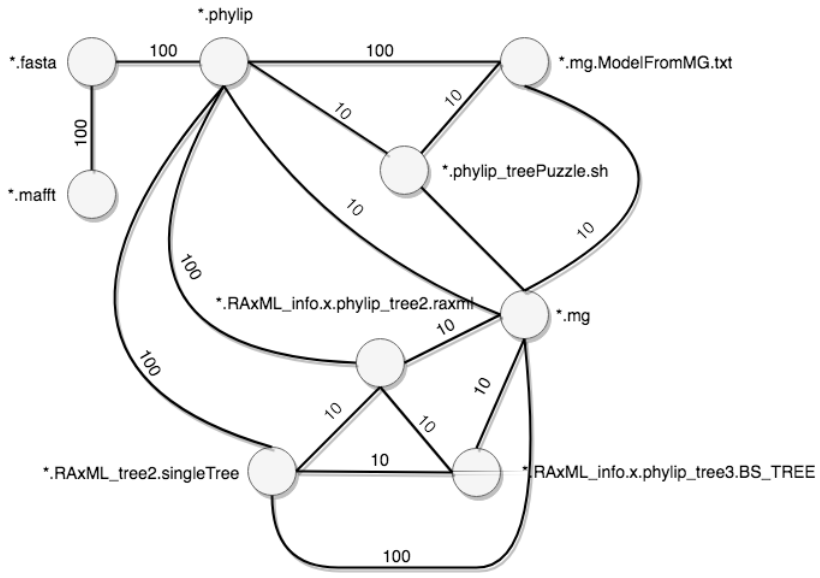

Figura 5.2: Um exemplo de um grafo de conflitos.

entre arquivos e, mesmo quando possível, é difícil relacionar o valor dessas penalidades com um custo propriamente (*i.e.*, em valores monetários). Além disso, a valoração dessas penalidades requer um trabalho grande do cientista especialista, que deve identificar e precificar cada uma das restrições entre arquivos, o que pode inviabilizar, em alguns casos, o processo como um todo. Sendo assim, considera-se a situação em que os arquivos consumidos/produzidos pelos experimentos nunca poderão ser armazenados juntos em um mesmo local (*buckets* ou serviços). Mesmo dispersando os arquivos em diferentes *buckets* de algum serviço de armazenamento em nuvem, um usuário mal intencionado que obtenha indevidamente as credenciais desse serviço, poderá ter acesso aos resultados desses experimentos. Portanto, visando minimizar esse risco, é interessante que esses arquivos conflitantes sejam distribuídos em *buckets* de diferentes serviços de armazenamento em nuvem.

Ao considerar diferentes serviços de armazenamento em nuvem, deve-se observar seus diferentes planos de contratação. Existem tipos de precificação dos mais variados nesses serviços de armazenamento em nuvem, divididos basicamente em: (i) um plano com custo fixo de contratação e disponibilidade em *gigabytes*, que não leva em consideração a quantidade de *bytes* que de fato está sendo armazenada, e (ii) um plano com custo variável, que cobra o cliente somente pela quantidade de *gigabytes* que estão de fato sendo usados, conhecido como "pague somente o que utilizar" (*"pay only for what you use"*). Vale ressaltar que, em ambos os casos, a disponibilidade de espaço é limitada.

Além disso, mesmo dentro de cada um desses planos de contratação de serviços, a

política de preços pode variar dependendo da quantidade de dados que o usuário armazena. O serviço de armazenamento em nuvem *iDrive*, por exemplo, tem custo igual a zero por até 5 *gigabytes* de arquivos armazenados, mas ao ultrapassar esse limite, existe a cobrança de $4.33 por mês para armazenar até 2 *terabytes*. Esse tipo de cobrança foi denominada de faixas de cobrança e são consideradas no cenário real do problema. Ademais, em alguns serviços existe um custo para transferência de arquivos, *i.e.*, uma espécie de taxa paga pelo cliente por cada *byte* de dados transferido para o servidor. Finalmente, a função objetivo visa minimizar o total desses custos envolvidos para contratar e utilizar o serviço de armazenamento em nuvem, respeitando não só a capacidade de cada serviço contratado, mas também os conflitos entre arquivos. A partir daqui, o problema será denominado Problema de Dispersão de Arquivos em Serviços de Armazenamento em Nuvem (PDASAN).

A prova de que o PDASAN abordado é NP-Difícil é direta, uma vez que esse problema é uma generalização do problema da mochila devido às restrições de capacidade [57], e admite o problema de *bin packing* com conflitos [18, 36, 54] como um caso especial.

5.2.1 Formulação Matemática para o PDASAN

Nesta seção, é descrita a formulação matemática para o problema PDASAN, introduzido neste capítulo. Considere um grafo não direcionado $G(N, E)$, onde $N = \{1, \ldots, |N|\}$ representa o conjunto de arquivos produzidos/consumidos por uma execução do *workflow*, e E o conjunto de arestas (i, j) que representam os conflitos entre os arquivos $i, j \in N$. Considere um conjunto de serviços de armazenamento em nuvem $K = \{1, \ldots, |K|\}$, $K = K_m \cup K_d$, onde K_m denota o subconjunto de serviços com a opção fixa de pagamento e K_d o subconjunto de serviços com a opção de pagamento que varia conforme a quantidade de arquivos a ele alocados. Considere L_k o conjunto de faixas de cobrança do serviço $k \in K$. Seja t_i o tamanho em *gigabytes* dos arquivos $i \in N$, e c_{kl} a capacidade de armazenamento do serviço $k \in K$ na faixa $l \in L_k$. Além disso, seja p_{kl} o custo de utilizar o serviço $k \in K$ na faixa $l \in L_k$ e f_k o custo de transferência cobrado para enviar dados para o servidor. As variáveis binárias x_{ik} e z_{kl} indicam, respectivamente, se o arquivo i está alocado ao serviço k e se o serviço k está sendo utilizado na faixa l. Seja q_{kl} a variável real que indica a soma do tamanho dos arquivos alocados no serviço k na faixa l. O problema consiste em encontrar o conjunto de alocações dos arquivos $i \in N$ no subconjunto de serviços $S \subseteq K$, que minimiza os custos de contratação e utilização dos serviços de armazenamento em nuvem, e respeite tanto a capacidade dos serviços contratados quanto os conflitos entre

arquivos.

O problema pode ser formulado como a seguir:

$$min \sum_{i=1}^{|N|} \sum_{k=1}^{|K|} f_k \cdot t_i \cdot x_{ik} + \sum_{k=1}^{|K_d|} \sum_{l=0}^{|L_k|} p_{kl} \cdot q_{kl} + \sum_{k=1}^{|K_m|} \sum_{l=0}^{|L_k|} p_{kl} \cdot z_{kl} \tag{5.1}$$

sujeito a:

$$\sum_{k=1}^{|K|} x_{ik} = 1, \forall i \in N \tag{5.2}$$

$$x_{ik} + x_{jk} \leq 1, \forall (i,j) \in E, \forall k \in K \tag{5.3}$$

$$\sum_{l=0}^{|L_k|} q_{kl} = \sum_{i=1}^{|N|} t_i \cdot x_{ik}, \forall k \in K \tag{5.4}$$

$$\sum_{i=1}^{|N|} t_i \cdot x_{ik} \leq \sum_{l=0}^{|L_k|} c_{kl} \cdot z_{kl}, \forall k \in K \tag{5.5}$$

$$\sum_{l=0}^{|L_k|} z_{kl} = 1, \forall k \in K \tag{5.6}$$

$$c_{k(l-1)} \cdot z_{kl)} \leq q_{kl} \leq c_{kl} \cdot z_{kl}, \forall k \in K, l \in L_k \setminus \{0\} \tag{5.7}$$

$$\sum_{i=1}^{|N|} x_{ik} \geq z_{kl}, \forall k \in K, l \in L_k \tag{5.8}$$

$$x_{ik} \in \{0,1\}, z_{kl} \in \{0,1\}, q_{kl} \in \mathbb{R}, \forall i,j \in N, \forall k \in K, \forall l \in L_k \tag{5.9}$$

A função objetivo (5.1) minimiza os custos de alocar todos os arquivos para um subconjunto de serviços, considerando tanto as políticas de preço fixo quanto as políticas em que o preço depende da quantidade de *gigabytes* que foram alocados aos serviços, bem como os custos de transferência dos arquivos. A Restrição (5.2) garante que cada arquivo deve ser alocado a exatamente um serviço de armazenamento em nuvem. A Restrição (5.3) assegura que os arquivos que possuem conflitos entre si não serão alocados ao mesmo

serviço. A Restrição (5.4) computa a soma do tamanho dos arquivos armazenados em um mesmo serviço. A Restrição (5.5) garante que, para cada serviço em sua respectiva faixa, sua capacidade máxima será respeitada. A Restrição (5.6) certifica que somente uma faixa será selecionada para cada serviço, enquanto que as Restrições (5.7) garantem que a soma do tamanho dos arquivos respeita a capacidade das faixas de cobrança. A Restrição (5.8) impõe que se existir um arquivo associado a um serviço, uma faixa desse serviço deve ser selecionada. A Restrição (5.9) define o domínio das variáveis. A Tabela 5.1 mostra um resumo das variáveis e dados de entrada do problema.

Tabela 5.1: Notação utilizada na formulação matemática do PDASAN.

Notação	**Descrição**
N	O conjunto de arquivos consumidos/produzidos pela execução do *workflow*
K_m	O subconjunto de serviços de armazenamento em nuvem que possuem custo fixo e contratação
K_d	O subconjunto de serviços de armazenamento em nuvem cujo custo de contratação depende dos arquivos lá armazenados
K	Conjunto total de serviços disponíveis $K = K_m \cup K_d$
L_k	Conjunto de faixas de cobrança do serviço k
t_i	Tamanho do arquivo i
c_{kl}	Capacidade do serviço k na faixa l
p_{kl}	Custo do serviço k na faixa l
f_k	Custo de transferência pago ao enviar um arquivo para o serviço k
x_{ik}	Variável binária que indica se o arquivo i está alocada ao serviço k
z_{kl}	Variável binária que indica se o serviço k está sendo usado na faixa l
q_{kl}	Variável real que indica a soma do tamanho dos arquivos alocados no serviço k na faixa l

5.3 Experimentos Computacionais

A fim de validar o modelo matemático proposto para o PDASAN, alguns experimentos foram conduzidos. Inicialmente, são apresentados o ambiente de execução, bem como as instâncias utilizadas. Em seguida, os resultados obtidos a partir da execução do modelo nessas instâncias são discutidos. Todos os dados e experimentos apresentados neste capítulo são públicos e estão disponíveis no repositório *OPTIC* do GitHub[6].

[6]https://github.com/UFFeScience/OPTIC

5.3.1 Ambiente de Testes

A formulação matemática detalhada anteriormente foi testada utilizando o *solver* IBM ILOG CPLEX versão 12.5.1. Todos os experimentos foram conduzidos em um PC com processador Intel®Core™ i5-5200 CPU @ 2.20GHz com 8GB RAM em ambiente Linux Ubuntu versão 16.04 LTS, com todas as opções de paralelismo desabilitadas. Os detalhes sobre as informações reais acerca dos serviços de internet que oferecem serviços de armazenagem em nuvem estão presentes na Tabela 5.2. Nessa tabela, a primeira coluna informa o nome do serviço, a segunda coluna indica se aquele serviço faz divisão em faixas de cobrança de sua política de preços (e capacidade de armazenamento), a terceira coluna informa se o serviço cobra preços fixos de contratação e, por fim, a última coluna indica se o serviço cobra taxas de transferência por arquivo enviado ao servidor.

Tabela 5.2: Lista de serviços de armazenamento em nuvem e suas principais características.

Serviço	**Faixas de Cobrança**	**Preço Fixo**	**Taxa de transferência**
Amazon	✓	✗	✓
Google Cloud	✗	✗	✓
Azure	✗	✗	✗
Mega	✓	✓	✗
SpiderOak	✓	✓	✗
IDrive	✓	✓	✗
pCloud	✓	✓	✗
sugarsync	✗	✓	✗
Dropbox	✗	✓	✗
Tresorit	✗	✓	✗
Mozy	✗	✓	✗
JustCloud	✗	✓	✗
Norton	✗	✓	✗
SafeCopy	✗	✓	✗
OpenDrive	✓	✓	✗
Zoolz	✗	✓	✗
ADrive	✗	✓	✗

5.3.2 Instâncias Baseadas em Workflows Reais

O *benchmark* de instâncias foi gerado utilizando informações reais coletadas a partir de execuções de alguns *workflows* públicos, a partir da página *Workflow Generator*[7]. Todas as restrições entre arquivos são previamente conhecidas e geradas por especialistas de cada *workflow*. Todos os *workflows* utilizados neste trabalho são descritos a seguir. *Montage* é

[7]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

um *workflow* de astronomia que coleta dados via telescópios e cria um mosaico de imagens a partir de objetos capturados no céu [95, 96]. O *LIGO* [1, 2] é um *workflow* utilizado para gerar e analisar as formas observadas em ondas gravitacionais durante a coalescência do espaço. *SIPHT* [61] é um *workflow* baseado em um projeto de bioinformática da Universidade de Harvard que busca RNAs não codificáveis (ncRNA) na base de dados NCBI. *Epigenomics* é um *workflow* criado pela USC Epigenome Center para trabalhar com sequenciamento do genoma humano [74]. Por último, *CyberShake* é um *workflow* comumente utilizado para predição e prevenção de terremotos no sul da Califónia [17].

5.3.3 Resultados Computacionais para PDASAN

Os experimentos computacionais do modelo matemático para o PDASAN são apresentados na Tabela 5.3. Nesses experimentos, o tempo máximo de execução do modelo foi de oito horas (28.800s). Na Tabela 5.3, para cada instância apresentada na primeira coluna, o número de arquivos consumidos/gerados por esse *workflow* está na segunda coluna e a terceira coluna reporta a solução ótima encontrada pelo modelo matemático, executando o resolvedor CPLEX. Caso esse resolvedor não encontre a solução ótima em oito horas (28.800 segundos), a melhor solução alcançada até o limite de tempo computacional é exibida. Caso o resolvedor não encontre nenhuma solução viável inteira, tem-se a indicação de um traço "-". O tempo computacional total é mostrado na quarta coluna. A última coluna indica quantos serviços foram necessários para o plano de distribuição de arquivos ótimo (de um total de 17 serviços, apresentados na última seção). O CPLEX foi executado com todas as opções padrões habilitadas, com exceção da opção de paralelismo, sendo executado em uma única *thread*.

Os resultados reportados pela Tabela 5.3 indicam que o CPLEX se comportou bem para praticamente todas as instâncias, produzindo boas soluções (quase sempre ótimas) em um tempo computacional considerado aceitável. Em alguns casos, no entanto, o *solver* CPLEX não encontrou a solução ótima, que foi o caso das instâncias SciPhy24731 e Epigenomics_997, considerando um tempo limite de 28800 segundos. Na instância SciPhy24731, o CPLEX não reportou nem mesmo uma solução inteira viável. Essas instâncias motivaram a proposta de uma solução heurística que pudesse, em um tempo computacional aceitável, encontrar uma solução viável para o problema abordado. Na Seção 5.4 a seguir, portanto, duas heurísticas baseadas nas metaheurísticas GRASP e *Simulated Annealing* são propostas como alternativas ao modelo matemático.

Tabela 5.3: Resultados Computacionais para o Modelo Matemático, usando o solver CPLEX, para o cenário real.

		CPLEX			
Instâncias	**|N|**	**Sol**	**Gap**	**Time (s)**	**#serv**
SciPhy21	21	35,36	0,00	0,04	13
SciPhy32	32	3,33	0,00	0,05	4
SciPhy102	102	5,83	0,00	0,72	6
SciPhy122	122	7,32	0,00	0,76	5
SciPhy1493	1493	771,37	0,00	4,67	12
SciPhy1615	1615	875,12	0,00	4,82	12
SciPhy16150	16150	15831,53	0,00	1909,29	15
SciPhy24731	24731	-	-	28800,00	-
Inspiral_30	163	70400,21	0,00	77,10	13
Inspiral_50	278	121444,00	0,00	8,60	13
Inspiral_100	546	237589,00	0,00	2,24	14
Inspiral_1000	5549	1658,38	0,00	55,48	12
Montage_25	134	43290,08	0,00	0,26	8
Montage_50	290	83072,88	0,00	1,44	12
Montage_100	618	152641,00	0,00	1,55	13
Montage_1000	6472	1490197,58	0,00	87,50	14
Sipht_30	1927	62329,10	0,00	28,67	15
Sipht_60	3976	124128,00	0,00	28,84	15
Sipht_100	6047	187709,00	0,00	364,04	15
Epigenomics_24	69	866196,00	0,00	0,04	5
Epigenomics_46	134	1483550,92	0,00	0,03	4
Epigenomics_100	297	10503100,00	0,00	0,25	10
Epigenomics_997	2969	131998648,23	10,70	28800,00	15
CyberShake_30	116	6698980,64	0,00	0,52	5
CyberShake_50	196	7239305,86	0,00	0,89	5
CyberShake_100	396	8186677,26	0,00	1,36	8
CyberShake_1000	4004	50175400,49	0,00	237,70	13

5.4 Abordagem Heurística

Em geral, métodos exatos tornam-se impraticáveis quando lidam com instâncias reais de grande porte. Nesses casos, uma boa alternativa é a utilização de heurísticas e metaheurísticas. Neste trabalho, duas heurísticas, uma baseada na metaheurística GRASP [30] e outra baseada na metaheurística *Simulated Annealing* [58] são propostas para resolver o PDASAN. A principal motivação é que essas heurísticas possam ser alternativas sempre que o modelo matemático discutido na Seção 5.3 não for capaz de encontrar uma solução inteira viável para o problema abordado, e.g., como aconteceu no caso da instância SciPhy24731. Ainda que o *solver* encontre uma solução viável para o problema, e.g., instância Epigenomics_997, é interessante que a abordagem exata não consuma tanto tempo

computacional quando comparado ao tempo de execução do *workflow*.

A heurística baseada em GRASP para Dispersão de Arquivos, que será denominada GRASP-DA, é composta por duas componentes, a saber: um algoritmo construtivo com características gulosas e aleatórias, e uma busca local. Cada uma dessas componente são detalhadas a seguir.

5.4.1 A Heurística Construtiva

A heurística construtiva proposta, denominada ConstMBS, tem por objetivo construir uma solução inicial viável para o problema, *i.e.*, um plano para dispersar os arquivos entre os serviços de armazenamentos em nuvem disponíveis. A ideia principal dessa heurística é que os arquivos com a menor quantidade de conflitos devem ser armazenados naqueles serviços cuja capacidade são maiores. Dessa forma, espera-se que a heurística possa alocar a maior quantidade possível de arquivos em um mesmo serviço, o que pode diminuir a ocorrência de uma situação inviável ao final da construção.

O pseudocódigo da heurística construtiva ConstMBS é apresentado pelo Algoritmo 14. Inicialmente, a solução s é inicializada como vazia, e são criadas duas listas, a primeira contendo todos os arquivos (LF) a serem distribuídos, e a segunda contendo todos os serviços de armazenamento em nuvem (LS). A lista de serviços é ordenada de maneira decrescente pela capacidade máxima de armazenamento, enquanto que a lista de arquivos é ordenada de maneira crescente pela quantidade de conflitos que os arquivos possuem – linhas 3 e 4. Então, para cada iteração do laço principal, uma lista restrita de arquivos candidatos a serem inseridos na solução é criada, baseada no parâmetro de entrada α, que funciona exatamente como o parâmetro da metaheurística GRASP [30], detalhado no Capítulo 2. Um arquivo candidato dessa lista restrita é escolhido de maneira aleatória para ser inserido em s – linhas 6 até 8. O arquivo selecionado é removido da lista de arquivos (linha 6) e será inserido no próximo serviço da lista de serviços cuja capacidade ainda não foi excedida e que não gere conflitos entre o arquivo selecionado e os demais arquivos do serviço em questão (linha 11). Os serviços são escolhidos de maneira sequencial e circular, reiniciando ao primeiro serviço da lista sempre que uma inserção de arquivo for bem sucedida. No caso de nenhum serviço disponível ser configurado como viável para inserção, *e.g.*, todos os serviços com a capacidade máxima preenchida ou com conflitos com todos os arquivos restantes, um procedimento para resolver conflitos é chamado *tentaResolverConflito* – linha 14.

O procedimento *tentaResolverConflito* (linha 14) busca por um serviço b em que o

arquivo f não possua conflito com nenhum outro arquivo do serviço b. Se esse serviço existir, a inviabilidade foi ocasionada exclusivamente pela restrição de capacidade. Assim, o procedimento procura por outro serviço b' que possua espaço suficiente para receber o arquivo f e todos os arquivos alocados ao serviço b, além de também mover todos os arquivo do serviço b' para o serviço b. Se esse serviço existe, a troca entre os serviços b' e b é feita. Caso contrário, a heurística construtiva termina e nenhuma solução viável é retornada.

Algoritmo 14 Heurística Construtiva ConstMBS

```
1: Entrada: Porcentagem de arquivos candidatos α
2: s ← ∅;
3: LF ← ordenaArquivos();
4: LS ← ordenaServiços();
5: enquanto LF não é vazio faça
6:    RCL ← criaLRC(LF, α);
7:    f ← obterArquivoAleatorio(RCL);
8:    LF ← remover(f, LF);
9:    k ← 0;
10:   enquanto f não foi inserido em k faça
11:      s ← tentaAlocar(f, k);
12:      k ← k+1;
13:      se não há nenhum serviço disponível então
14:         tentaResolverConflito(s,f);
15:      fim se
16:   fim enquanto
17: fim enquanto
18: retorne s;
```

Existem casos em que a solução produzida ao final da heurística construtiva é inviável, mesmo após a aplicação do procedimento *tentaResolverConflito*. Situações de inviabilidade como essas podem ocorrer em casos em que a quantidade de conflitos é tão grande que a quantidade de serviços não é suficiente, ou em casos em que a soma do tamanho dos arquivos é maior que a capacidade dos serviços. Em ambos os casos, a heurística proposta nunca será capaz de produzir uma solução viável devido às características das instâncias. Embora exista a chance de situações como essas ocorrerem em um cenário real (considerando *workflows* científicos que consumem/produzem imensas quantidades de dados), a heurística proposta encontrou soluções viáveis para todas as instâncias do *benchmark* analisado.

5.4.2 Busca Local

Ainda que a heurística construtiva seja capaz de sempre encontrar uma boa solução, retornando um plano viável para dispersar os arquivos em vários serviços em nuvem, considera-se importante tentar melhorar essa solução inicial. Nesta seção, é descrito um procedimento simples, baseado no movimento de realocação, que é aplicado com o objetivo de melhorar uma solução construída inicialmente. O movimento de realocação consiste em transferir um arquivo de um serviço de armazenamento em nuvem para outro, buscando sempre movimentos de melhoria. Essa vizinhança é explorada exaustivamente até que todos os movimentos de melhoria sejam aplicados e um ótimo local seja alcançado. A implementação da vizinhança de realocação foi feita utilizando a estratégia de primeiro aprimorante. O Algoritmo 15 detalha os passos dessa vizinhança.

Algoritmo 15 Heurística Relocate

Entrada: Solução s;

```
1:  s' ← s;
2:  melhorou ← true;
3:  enquanto melhorou faça
4:     melhorou ← false;
5:     para todo arquivo f ∈ N  faça
6:        remCost ← calcular o custo parcial de remover arquivo f de s';
7:        para todo serviço k ∈ K faça
8:           addCost ← calcular o custo parcial de inserir arquivo f em k;
9:           se remCost > addCost então
10:             s' ← realocar(f, k);
11:             melhorou ← true;
12:             break;
13:          fim se
14:       fim para
15:    fim para
16: fim enquanto
17: retorne s';
```

O procedimento de realocação inicia com uma solução s e, para cada arquivo f na lista de arquivos, calcula-se o custo de remover o arquivo f da solução s corrente (linha 6). Em seguida, para cada serviço k na lista de serviços que não possuem conflito com f e capacidade suficiente para inseri-lo, calcula-se o custo de inserir f em k (linha 8). Sempre que o custo de remover o arquivo f da solução for maior que o custo de inseri-lo em outro serviço (linha 9), uma solução melhor é encontrada e portanto a realocação é realizada. Após a realocação, o valor da variável *melhorou* é atualizado para *true* e a busca continua (linhas 10 e 11). A aplicação do movimento de realocação é feita até que nenhuma melhoria

de solução seja encontrada.

Assim, o pseudocódigo da heurística GRASP-DA proposta é apresentado no Algoritmo 16. O GRASP-DA é executado por *maxIter* iterações. Nos testes, o valor de *maxIter* estipulado foi de 50 iterações. Valores maiores que 50 iterações apenas consumiram mais tempo computacional, não resultando em grandes melhorias na qualidade das soluções.

Algoritmo 16: GRASP para o PDASAN

```
1: GRASP-DA(α,maxIter)
2: s* ← s ← ∅; f(s*) ← ∞
3: para iter = 1 até maxIter faça
4:    s ← ConstMBS(α)
5:    s ← Relocate(s)
6:    se f(s) < f(s*) então
7:       s* ← s
8:    fim se
9: fim para
10: retorne s*
```

5.4.3 Simulated Annealing para o PDASAN

Além da heurística baseada em GRASP, outra abordagem heurística desenvolvida neste trabalho foi baseada na metaheurística *Simulated Annealing* (SA). Para tanto, utilizou-se a componente construtiva e uma pequena variação da busca local do GRASP, detalhadas anteriormente nas Seções 5.4.1 e 5.4.2, como base para a heurística SA. Essa heurística será denominada SA-DA.

Inicialmente, como o construtivo do GRASP não garante que uma solução viável seja sempre retornada, a etapa de construção é repetida até que uma solução viável seja encontrada para se tornar a solução inicial do SA. No laço de movimentação do SA, movimentos da vizinhança *Relocate* são selecionados aleatoriamente e aplicados à solução atual. A cada movimento aplicado dessa vizinhança, a aceitação da nova solução s' está sujeita ao critério de aceitação probabilístico do SA. Se o movimento gerar uma melhoria na solução atual, ela é aceita. Caso contrário, o movimento pode ser aceito se um número aleatório r satisfazer $r < e^{\frac{-\Delta(s,s')}{T}}$, sendo T a temperatura corrente e $\Delta(s, s')$ a diferença de custo entre as soluções s e s'.

O pseudocódigo da heurística SA-DA pode ser visualizado no Algoritmo 17. Inicial-

mente, uma solução viável é construída pelo construtivo ConstMBS (linhas 2 a 4). Em seguida, após inicializar a temperatura (linha 6), vários movimentos são aplicados à solução corrente s, por meio da vizinha Relocate(s) (linha 11). Se esse movimento fornecer uma solução s' com custo melhor que o custo de s, a solução s' é aceita. Caso contrário, s' ainda pode ser aceita, mas depende da probabilidade calculada na linha 15. Após o laço de aplicação movimentos, aplica-se a taxa de resfriamento para atualizar a temperatura (linha 18) e a melhor solução da heurística é armazenada.

Algoritmo 17 Simulated Annealing para o PDASAN

1: **SA-DA** (α, T_0, T_c, ω, SA_{max})
2: **repita**
3: $s \leftarrow$ ConstMBS(α);
4: **até** solução s ser viável.
5: $s^* \leftarrow s$;
6: $T \leftarrow T_0$;
7: **enquanto** $T > T_c$ **faça**
8: $iter \leftarrow 0$;
9: **enquanto** $iter < SA_{max}$ **faça**
10: $iter \leftarrow iter + 1$;
11: $s' \leftarrow$ Relocate(s);
12: **se** $f(s') < f(s)$ **então**
13: $s \leftarrow s'$;
14: **senão**
15: $s \leftarrow s'$, com probabilidade $e^{\frac{-(f(s)-f(s'))}{T}}$;
16: **fim se**
17: **fim enquanto**
18: $T \leftarrow \omega T$;
19: $s^* \leftarrow min(s^*, s)$;
20: **fim enquanto**
21: **retorne** s^*;

5.4.4 Resultados Computacionais para as Heurísticas GRASP-DA e SA-DA

Nesta seção, são apresentados os resultados computacionais alcançados pelas heurísticas propostas, denominadas, respectivamente, GRASP-DA e SA-DA. Ambas as heurísticas foram implementadas em C++, usando o compilador g++ versão 4.8.2. O ambiente computacional é o mesmo daquele apresentado na Seção 5.3.1. As heurísticas GRASP-DA e SA-DA foram executadas dez vezes para cada instância, cada execução com uma semente aleatória diferente. O valor do parâmetro de entrada α, definido em experimentos preliminares, foi $0,9$. Outros valores geravam, em algumas instâncias, diversas soluções

inviáveis na heurística construtiva. Com esse valor, foi possível construir soluções viáveis para todas as instâncias do problema. Para o SA-DA, os valores dos parâmetros α, T_0, T_c, ω, SA_{max} utilizados foram, respectivamente, 0,9, 10000, 0,00001, 0,975, e 2000. Esses valores foram definidos em experimentos preliminares.

A Tabela 5.4 reporta os resultados dos experimentos computacionais da heurística GRASP-DA, conduzidos no mesmo *benchmark* em que o modelo de programação inteira foi avaliado. A primeira e a segunda colunas indicam o nome da instância e a quantidade de arquivos. A terceira coluna reporta, para cada instância, o custo da melhor solução encontrada dentre todas as execuções. A quarta e a quinta coluna, apresentam, respectivamente, a média de qualidade de solução e de tempo. Finalmente, a última coluna informa a quantidade média de serviços utilizados para a dispersão dos arquivos, considerando as melhores soluções de cada execução.

De acordo com os resultados reportados na Tabela 5.4, é possível afirmar que a heurística GRASP-DA obteve um desempenho promissor para todas as instâncias, encontrando soluções em um tempo computacional aceitável. Apesar de simples, a heurística proposta ainda conseguiu reportar soluções viáveis para instância SciPhy24731, caso para o qual o modelo matemático, implementado com o *solver* CPLEX, não foi capaz de encontrar nenhuma solução viável, mesmo utilizando oito horas de processamento. Além disso, para a instância Epigenomics_997, a heurística proposta encontrou soluções melhores que a solução reportada pelo CPLEX, em um tempo computacional muito inferior. Esses resultados indicam que o GRASP-DA pode funcionar como uma alternativa para solucionar o problema PDASAN, especialmente nos casos em que o CPLEX não consegue retornar uma solução viável em um tempo computacional aceitável.

A Tabela 5.5 apresenta os resultados obtidos pela heurística SA-DA. A comparação com o GRASP-DA está presente na Tabela 5.6. Ambas as tabelas seguem a mesma estrutura da Tabela 5.4, com exceção da coluna `Avg#serv` que não é exibida na Tabela 5.6. Analisando a Tabela 5.5, é possível perceber que a heurística SA-DA, assim como o GRASP-DA, obteve um desempenho promissor, conseguindo alcançar soluções em um tempo computacional aceitável. Novamente, soluções viáveis foram encontradas para a instância SciPhy24731 e, para a instância Epigenomics_997, a heurística proposta também encontrou soluções melhores que a solução reportada pelo CPLEX, em um tempo computacional muito inferior.

Na comparação entre as duas heurísticas GRASP-DA e SA-DA, presente na Tabela 5.6, é possível notar que o desempenho da heurística SA-DA foi superior ao do

Tabela 5.4: Resultados computacionais para a heurística GRASP-DA em instâncias do PDASAN.

		GRASP-DA			
Instâncias	**|N|**	**Melhor**	**Média**	**Tempo (s)**	**Avg #serv**
SciPhy21	21	39,52	39,52	0,00	13,0
SciPhy32	32	11,32	11,32	0,02	6,0
SciPhy102	102	8,33	10,94	0,13	5,0
SciPhy122	122	10,06	10,59	0,19	5,0
SciPhy1493	1493	841,57	843,14	13,68	15,5
SciPhy1615	1615	947,95	951,41	16,04	15,2
SciPhy16150	16150	18778,20	18804,16	648,84	17,0
SciPhy24731	24731	30168,73	30304,33	1700,56	17,0
Inspiral_30	163	71036,67	71036,67	0,08	10,0
Inspiral_50	278	122055,00	122055,00	0,21	11,0
Inspiral_100	546	238194,84	238194,93	0,70	11,0
Inspiral_1000	5549	2364,08	2364,61	120,73	6,0
Montage_25	134	43425,15	43436,85	0,12	8,8
Montage_50	290	83504,60	83504,60	0,56	12,0
Montage_100	618	153118,66	153118,66	2,13	12,0
Montage_1000	6472	1490528,63	1490861,24	226,48	12,0
Sipht_30	1927	81247,16	81247,16	8,05	15,0
Sipht_60	3976	160478,86	160478,86	34,16	16,0
Sipht_100	6047	243073,00	243073,19	79,65	16,0
Epigenomics_24	69	877995,00	877995,00	0,01	4,0
Epigenomics_46	134	1500268,13	1500268,13	0,03	5,0
Epigenomics_100	297	10510770,00	10510770,00	0,15	7,0
Epigenomics_997	2969	124331432,00	125884247,20	13,94	13,1
CyberShake_30	116	6699338,50	6699338,50	0,11	2,0
CyberShake_50	196	7239560,00	7239560,00	0,29	3,0
CyberShake_100	396	8187138,00	8187138,00	1,22	3,0
CyberShake_1000	4004	50176400,00	50176400,00	89,65	9,0

GRASP-DA, encontrando soluções melhores em um menor tempo computacional. No geral, a heurística SA-DA obteve 22 melhores soluções, enquanto o GRASP-DA obteve apenas cinco. Em relação à média de custo de solução, a heurística SA-DA obteve 17 melhores médias, enquanto o GRASP-DA obteve 10 melhores médias. Finalmente, em relação à comparação do tempo médio das heurísticas, o SA-DA foi muito superior ao GRASP-DA, obtendo em poucos segundos, na maioria das vezes, soluções iguais ou até melhores que as obtidas pelo GRASP-DA.

Outra análise foi conduzida para verificar se os ganhos referentes à média de solução reportados na Tabela 5.6 têm significância estatística, realizando o teste estatístico não paramétrico de *Wilcoxon* pareado e unicaudal [101]. Esse teste é geralmente aplicado para avaliar dois algoritmos que possuem componentes aleatórias, e identificar se a diferença entre as médias de resultados obtidos por esses algoritmos foi, de fato, devido à superioridade de algum deles ou simplesmente devido à aleatoriedade dos métodos [101]. Para realizar o teste, foi considerado p-valor igual a $0,05$. Após a análise dos resultados, comparando as médias de soluções alcançadas pelos algoritmos GRASP-DA e SA-DA, é possível afirmar que, para todas as instâncias, houve diferença, com significância estatística, entre os valores de soluções alcançados pelas duas heurísticas. Somente em dois casos não houve significância estatística: a instância Inspiral_100 e Montage_25. Em ambos os casos, a média do GRASP-DA havia sido melhor. Em resumo, o GRASP-DA obteve 10, melhores médias, mas apenas 8 com significância estatística. Já o SA-DA obteve 17 melhores médias, todas com significância estatística. Isso indica que a probabilidade de a diferença de desempenho entre os algoritmos ter sido devido à aleatoriedade é menor que 5%.

Para melhor visualizar as soluções reportadas pelas heurísticas apresentadas neste capítulo, apresenta-se de maneira visual o plano de dispersão dos arquivos nos variados serviços de armazenamento em nuvem nas Figuras 5.3 e 5.4. Nessas figuras, em que são observadas soluções reais para as instâncias SciPhy_21 e SciPhy_32, cada arquivo é representado por um nó do grafo e as arestas representam os conflitos existentes entre os arquivos. As cores dos nós representam o serviço para o qual aquele arquivo será enviado quando aquele plano de dispersão for utilizado.

Tabela 5.5: Resultados computacionais para a heurística SA-DA em instâncias do PDA-SAN.

		SA-DA			
Instâncias	**|N|**	**Melhor**	**Média**	**Tempo (s)**	**Avg #serv**
SciPhy21	21	35,36	35,36	0,16	13,0
SciPhy32	32	5,83	6,89	0,19	6,0
SciPhy102	102	35,07	59,26	0,19	12,0
SciPhy122	122	54,50	71,48	0,20	12,8
SciPhy1493	1493	880,00	901,52	0,75	16,0
SciPhy1615	1615	932,18	1001,72	0,80	16,1
SciPhy16150	16150	17413,71	17512,39	8,74	17,0
SciPhy24731	24731	29799,21	30016,79	99,38	17,0
Inspiral_30	163	70933,29	70992,21	0,15	11,1
Inspiral_50	278	121963,73	122033,72	0,15	12,2
Inspiral_100	546	238114,92	238971,80	0,17	13,5
Inspiral_1000	5549	2045,17	2045,94	2,50	16,8
Montage_25	134	43437,09	43441,52	0,17	9,0
Montage_50	290	83486,74	86709,18	0,19	12,8
Montage_100	618	153117,66	160141,05	0,25	16,6
Montage_1000	6472	1490621,38	1557985,96	2,96	17,0
Sipht_30	1927	63715,43	66420,42	0,30	16,7
Sipht_60	3976	127156,51	131582,11	0,62	16,9
Sipht_100	6047	194520,53	199617,41	1,10	17,0
Epigenomics_24	69	866376,25	867207,06	0,11	4,2
Epigenomics_46	134	1484779,63	1485889,79	0,11	6,0
Epigenomics_100	297	10504144,00	10504502,00	0,12	7,1
Epigenomics_997	2969	119301072,00	119698168,00	0,32	14,1
CyberShake_30	116	6699285,00	6699300,55	0,19	11,9
CyberShake_50	196	7239497,00	7239520,75	0,20	14,9
CyberShake_100	396	8187015,00	8187048,90	0,25	16,5
CyberShake_1000	4004	50176356,00	50176796,80	1,10	17,0

Tabela 5.6: Resultados computacionais para as heurísticas GRASP-DA e SA-DA em instâncias do PDASAN.

		GRASP-DA			**SA-DA**		
Instâncias	**|N|**	**Melhor**	**Média**	**Tempo (s)**	**Melhor**	**Média**	**Tempo (s)**
SciPhy21	21	39,52	39,52	**0,00**	**35,36**	**35,36**	0,16
SciPhy32	32	11,32	11,32	**0,02**	**5,83**	**6,89**	0,19
SciPhy102	102	**8,33**	**10,94**	**0,13**	35,07	59,26	0,19
SciPhy122	122	**10,06**	**10,59**	**0,19**	54,50	71,48	0,20
SciPhy1493	1493	**841,57**	**843,14**	13,68	880,00	901,52	**0,75**
SciPhy1615	1615	947,95	**951,41**	16,04	**932,18**	1001,72	**0,80**
SciPhy16150	16150	18778,20	18804,16	648,84	**17413,71**	**17512,39**	**8,74**
SciPhy24731	24731	30168,73	30304,33	1700,56	**29799,21**	**30016,79**	**99,38**
Inspiral_30	163	71036,67	71036,67	**0,08**	**70933,29**	**70992,21**	0,15
Inspiral_50	278	122055,00	122055,00	0,21	**121963,73**	**122033,72**	**0,15**
Inspiral_100	546	238194,84	**238194,93**	0,70	**238114,92**	238971,80	**0,17**
Inspiral_1000	5549	2364,08	2364,61	120,73	**2045,17**	**2045,94**	**2,50**
Montage_25	134	**43425,15**	**43436,85**	**0,12**	43437,09	43441,52	0,17
Montage_50	290	83504,60	**83504,60**	0,56	**83486,74**	86709,18	**0,19**
Montage_100	618	153118,66	**153118,66**	2,13	**153117,66**	160141,05	**0,25**
Montage_1000	6472	**1490528,63**	**1490861,24**	226,48	1490621,38	1557985,96	**2,96**
Sipht_30	1927	81247,16	81247,16	8,05	**63715,43**	**66420,42**	**0,30**
Sipht_60	3976	160478,86	160478,86	34,16	**127156,51**	**131582,11**	**0,62**
Sipht_100	6047	243073,00	243073,19	79,65	**194520,53**	**199617,41**	**1,10**
Epigenomics_24	69	877995,00	877995,00	**0,01**	**866376,25**	**867207,06**	0,11
Epigenomics_46	134	1500268,13	1500268,13	**0,03**	**1484779,63**	**1485889,79**	0,11
Epigenomics_100	297	10510770,00	10510770,00	0,15	**10504144,00**	**10504502,00**	**0,12**
Epigenomics_997	2969	124331432,00	125884247,20	13,94	**119301072,00**	**119698168,00**	**0,32**
CyberShake_30	116	6699338,50	6699338,50	**0,11**	**6699285,00**	**6699300,55**	0,19
CyberShake_50	196	7239560,00	7239560,00	0,29	**7239497,00**	**7239520,75**	**0,20**
CyberShake_100	396	8187138,00	8187138,00	1,22	**8187015,00**	**8187048,90**	**0,25**
CyberShake_1000	4004	50176400,00	**50176400,00**	89,65	**50176356,00**	50176796,80	**1,10**

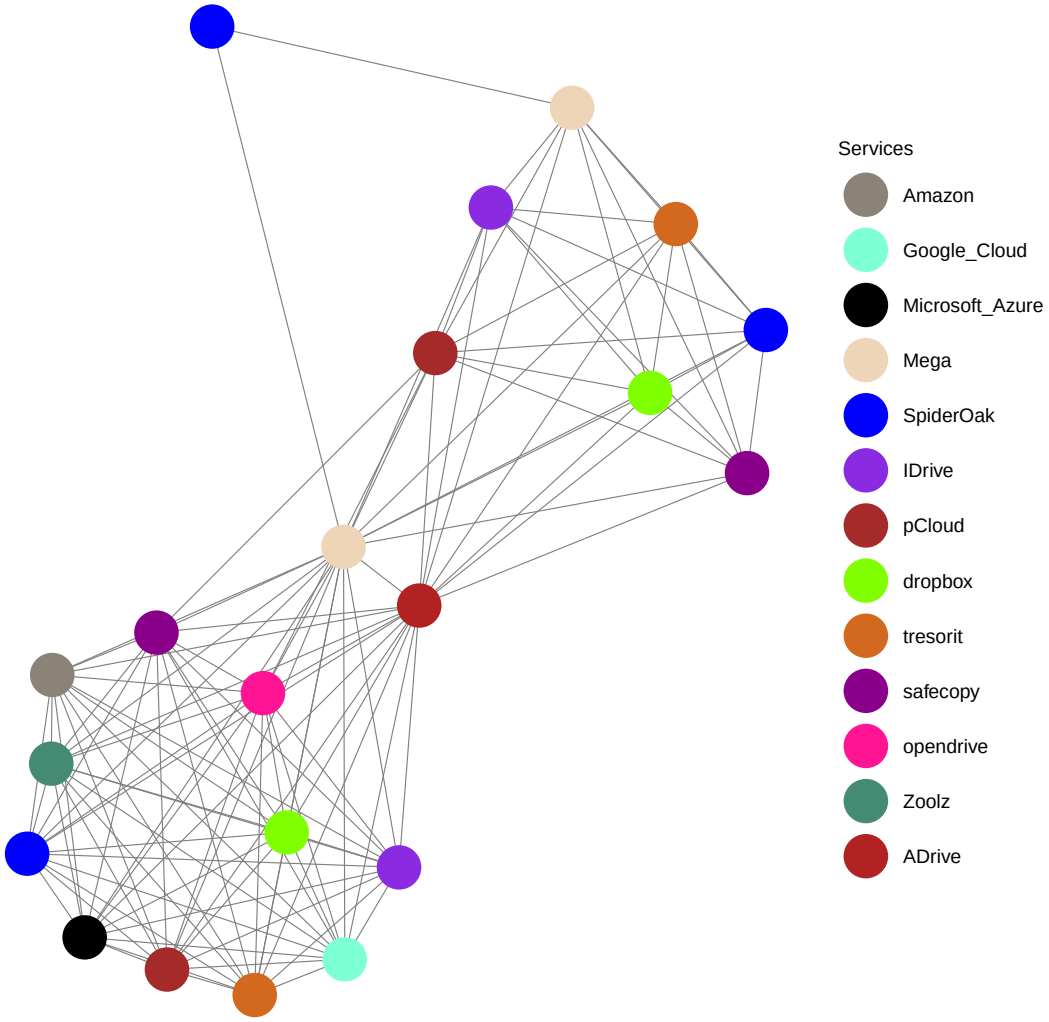

Figura 5.3: Solução com o plano de distribuição de arquivos para a instância SciPhy21.

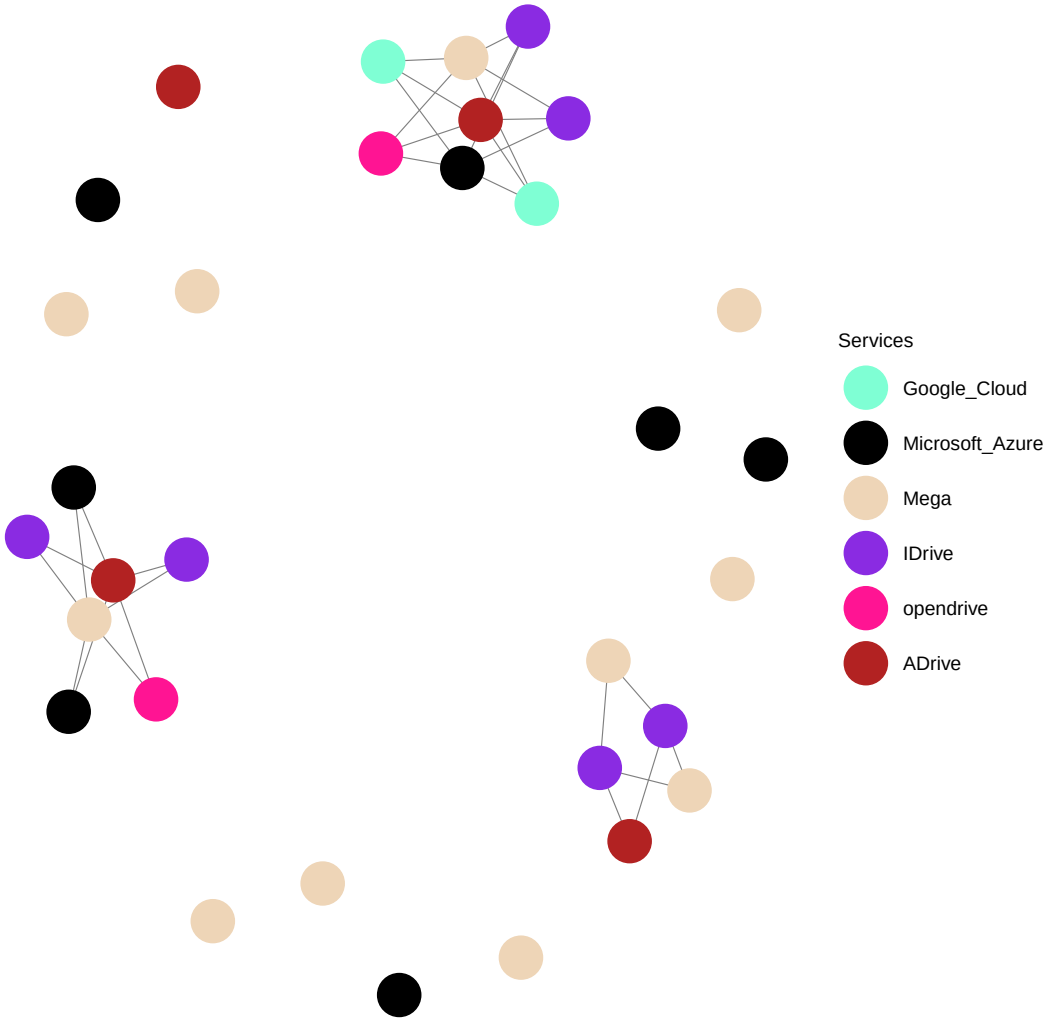

Figura 5.4: Solução com o plano de distribuição de arquivos para a instância SciPhy32.

Capítulo 6

Conclusões e Trabalhos Futuros

A hibridização de heurísticas multipartida (em especial, aquelas baseadas na metaheurística GRASP) com técnicas de mineração de dados foi amplamente investigada nos últimos dez anos [68]. Essa combinação, que obteve ótimos resultados em diversos problemas de otimização combinatória, caracteriza-se por extrair padrões de um conjunto elite de soluções construído ao longo de iterações iniciais da heurística base. A ideia principal é utilizar os padrões minerados para construir soluções de melhor qualidade e, assim, guiar a busca no espaço de soluções.

Embora essa combinação tenha alcançado sucesso até o momento, o escopo de utilização dessas abordagens é, ainda, limitado. Isso deve ao fato de essa hibridização ter sido explorada somente em heurísticas multipartidas, tais como: (i) GRASP [83], (ii) GRASP com Reconexão por caminhos [11], (iii) GRASP com *Variable Neighborhood Descent* [44], (iv) Heurísticas multipartida [67] e (v) Heurística multipartida com *Iterated Local Search* [64]. Além disso, em geral, a aplicação de mineração de dados é restrita a problemas cuja representação da solução é caracterizada por um conjunto de elementos e, portanto, não leva em consideração a ordem (ou sequência) dos elementos na solução, com exceção dos trabalhos de Guerine *et al.* [44] e Maia *et al.* [64].

Nesta Tese, buscou-se ampliar o espoco da inserção de técnicas de mineração de dados para auxiliar na busca heurística. Assim, essa hibridização foi investigada em outra metaheurística que possui estrutura completamente diferente da estrutura multipartida do GRASP. O estudo se deu com a metaheurística *Simulated Annealing*, aplicada a dois diferentes problemas de otimização combinatória. Para o primeiro, o problema de rotulação cartográfica de pontos, descrito no Capítulo 3, os resultados alcançados foram claramente superiores. Uma nova maneira de utilizar os padrões minerados foi proposta, adaptada à estrutura do SA, conseguindo explorar melhor o espaço de busca da heurística SA e,

consequentemente, reportando soluções melhores que a heurística original. Além disso, para várias instâncias, foram reportadas melhores soluções que as conhecidas na literatura do problema. No segundo problema, o problema de agrupamento capacitado com centro geométrico, descrito no Capítulo 4, inicialmente foi discutido uma nova maneira de construir a base de dados de solução e também de utilizar os padrões minerados, uma vez que o problema abordado não possuía as mesmas características que os demais problemas em que a abordagem híbrida com mineração de dados foi examinada (*i.e.*, a solução do problema não era caracterizada por um conjunto de elementos, e sim um conjunto de conjuntos). Os resultados computacionais apontaram uma leve melhora na qualidade de solução no momento em que os padrões foram utilizados. Embora essa melhora não tenha sido devidamente constatada como ocorreu no primeiro problema abordado, experimentos complementares indicaram que os padrões utilizados em heurísticas baseadas em SA puderam beneficiá-la, principalmente quando há pelo menos dois resfriamentos.

Além disso, esta Tese também explorou, utilizando uma heurística baseada em SA, um problema real de otimização combinatória, motivado pela demanda de um grupo de pesquisa que trabalha com *workflows* científicos, liderado pelo Professor Daniel de Oliveira (IC-UFF). Em seu trabalho, a constante manipulação de dados sensíveis/sigilosos gerados a partir de *workflows* científicos, executados em serviços em nuvem, faz com que exista a preocupação com a privacidade e confidencialidade desses dados, que não devem ser acessados por pessoas não autorizadas. Assim, visando minimizar esse risco, foi proposto a criação de um plano de dispersão dos arquivos gerados/consumidos por *workflows* científicos, evitando que arquivos que tenham qualquer relação semântica ou estrutural (previamente indicadas por um especialista da área) possam ser armazenados juntos. Uma modelagem matemática foi proposta utilizando programação linear inteira para descrever o problema, e o desempenho do modelo proposto foi avaliado em diversas instâncias reais, obtidas a partir de execuções de *workflows* públicos da literatura. Nos experimentos com o modelo proposto, foi possível perceber um bom desempenho da abordagem, alcançando soluções ótimas para praticamente todas as instâncias do *benchmark*, em um tempo computacional aceitável. Somente em dois casos o modelo não conseguiu encontrar as soluções ótimas, mesmo utilizando oito horas de processamento. Desses, em apenas um caso o modelo não encontrou nenhuma solução inteira viável. Para resolver as instâncias em que o modelo não conseguiu reportar soluções ótimas, foram propostas duas heurísticas, baseadas nas metaheurísticas GRASP e SA. As comparações feitas a partir de vários experimentos indicaram que tanto a heurística baseada em GRASP, quanto a baseada em SA, obtiveram resultados eficazes, conseguindo reportar soluções viáveis para

a instância em que o modelo não conseguiu resolver. Na comparação entre as duas heurísticas propostas, a baseada em SA foi superior à baseada em GRASP, obtendo os melhores resultados tanto em relação à qualidade das soluções encontradas, quanto em relação ao tempo computacional médio.

Como trabalhos futuros, pretende-se avaliar a combinação de mineração de dados com outras heurísticas baseadas em *Simulated Annealing*, investigando a relação entre a quantidade de resfriamentos realizados nessas heurísticas com o benefício da aplicação dos padrões ao longo de vários resfriamentos. Outra possibilidade de trabalho futuro é explorar a combinação de mineração de dados com metaheurísticas populacionais, tais como algoritmos genéticos e suas variações (e.g., BRKGA), seguindo a linha do trabalho de Santos *et al.* [98]. Em relação ao problema proposto para dispersão de arquivos em serviços de armazenamento em nuvem, pretende-se pesquisar novas possibilidades para melhorar o modelo, adaptando cortes de problemas similares ao modelo proposto nesta Tese. Para as abordagens heurísticas propostas (GRASP-DA e SA-DA), podem ser implementadas novas estruturas de vizinhança, em especial as que trabalham com busca em vizinhança larga, como as reportadas no trabalho de Capua *et al.* [18]. Além disso, também é possível explorar a combinação dessas heurísticas com mineração de dados a fim de tornar essas heurísticas ainda mais eficientes.

Ao longo do desenvolvimento desta tese, os seguintes artigos foram publicados ou submetidos. O primeiro artigo é parte do estudo apresentado no Capítulo 3. O segundo artigo fez parte do estudo apresentado no Capítulo 4. O terceiro artigo listado reuniu todos os experimentos realizados com o problema de rotulação cartográfica de pontos. Por fim, o último artigo listado é o resultado do trabalho apresentado no Capítulo 5, em pesquisa realizada em conjunto com outros pesquisadores da UFF, do LNCC e da UFRJ. Os dois primeiros artigos já foram publicados, enquanto que os dois últimos foram submetidos às respectivas revistas indicadas em cada item, e ambos já passaram por uma fase de revisão.

- Guerine, M.; Rosseti, I.; Plastino, A. Uma Metaheurística Híbrida com Mineração de Dados para o Problema de Rotulação Cartográfica de Pontos. Simpósio Brasileiro de Pesquisa Operacional (Porto de Galinhas, PE, Brasil, 2015), pp. 2150-2161.

- Guerine, M.; Stockinger, M.; Rosseti, I.; Plastino, A. Heurística Híbrida com Mineração de Dados para o Problema de Agrupamento Capacitado com Centro Geométrico. Simpósio Brasileiro de Pesquisa Operacional (Blumenau, SC, Brasil, 2017), pp. 1933-1944.

- Guerine, M.; Rosseti, I.; Plastino, A. *A hybrid data mining heuristic to solve the point-feature cartographic label placement problem.* Submetido para o journal *International Transactions in Operational Research* em 2018.

- Guerine, M.; Stockinger, M.; Rosseti, I.; Simonetti, L.; Ocaña, Kary.; de Oliveira, D.; Plastino, A. *A Provenance-based Heuristic for Preserving Results Confidentiality in Cloud-based Scientific Workflows.* Submetido para o journal *Future Generations Computer Systems* em 2018.

Referências

[1] Abbott, B.; Abbott, R.; Abbott, T.; Acernese, F.; Ackley, K.; Adams, C.; Adams, T.; Addesso, P.; Adhikari, R.; Adya, V., et al. Upper limits on gravitational waves from scorpius x-1 from a model-based cross-correlation search in advanced ligo data. *The Astrophysical Journal 847*, 1 (2017), 47.

[2] Abbott, B. P.; Abbott, R.; Abbott, T.; Acernese, F.; Ackley, K.; Adams, C.; Adams, T.; Addesso, P.; Adhikari, R.; Adya, V., et al. Search for intermediate mass black hole binaries in the first observing run of advanced ligo. *Physical Review D 96*, 2 (2017), 022001.

[3] Abramson, D.; Krishnamoorthy, M.; Dang, H. Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research 16* (1998).

[4] Afgan, E.; Baker, D.; Chilton, J.; Coraor, N.; Team, T. G.; Taylor, J. Galaxy cluster to cloud - genomics at scale. In *Proceedings of the 9th Gateway Computing Environments Workshop* (Piscataway, NJ, USA, 2014), GCE '14, IEEE Press, pp. 47–50.

[5] Agrawal, R.; Imieliński, T.; Swami, A. Mining association rules between sets of items in large databases. *SIGMOD Record 22* (1993), 207–216.

[6] Agrawal, R.; Srikant, R. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases* (1994), pp. 487–499.

[7] Aiex, R. M.; Resende, M. G. C.; Ribeiro, C. C. `TTT plots`: A perl program to create time-to-target plots. *Optimization Letters 1* (2007), 355–366.

[8] Alvim, A. C. F.; Taillard, É. D. POPMUSIC for the point-feature label placement problem. *European Journal of Operational Research 192* (2009), 396–413.

[9] And, A. P. Mining frequent itemsets using patricia tries, 2003.

[10] Aragon, C. R.; Runge, K. J. Workflow management for high volume supernova search. In *Proceedings of the 2009 ACM Symposium on Applied Computing* (New York, NY, USA, 2009), SAC '09, ACM, pp. 949–955.

[11] Barbalho, H.; Rosseti, I.; Martins, S. L.; Plastino, A. A hybrid data mining GRASP with path-relinking. *Computers & Operations Research 40* (2013), 3159–3173.

[12] Berger, D.; Gendron, B.; yves Potvin, J.; Raghavan, S.; Soriano, P. Tabu search for a network loading problem with multiple facilities. *Journal of Heuristics 6* (2000), 253–267.

[13] Bharathi, S.; Chervenak, A.; Deelman, E.; Mehta, G.; Su, M. H.; Vahi, K. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science* (Nov 2008), pp. 1–10.

[14] Blum, C.; Aguilera, M. J. B.; Roli, A.; Sampels, M. *Hybrid Metaheuristics: An Emerging Approach to Optimization*. Springer Publishing Company, Incorporated, 2008.

[15] Branco, Jr., E. C.; Monteiro, J. M.; de C. e Silva, R. R.; Machado, J. C. A new approach to preserving data confidentiality in the cloud. In *Proceedings of the 20th International Database Engineering; Applications Symposium* (New York, NY, USA, 2016), IDEAS '16, ACM, pp. 256–263.

[16] Bryant, R. E. Data-intensive scalable computing for scientific applications. *Computing in Science Engineering 13*, 6 (Nov 2011), 25–33.

[17] Callaghan, S.; Maechling, P.; Deelman, E.; Vahi, K.; Mehta, G.; Juve, G.; Milner, K.; Graves, R.; Field, E.; Okaya, D.; Gunter, D.; Beattie, K.; Jordan, T. H. Reducing time-to-solution using distributed high-throughput mega-workflows - experiences from SCEC cybershake. In *Fourth International Conference on e-Science, e-Science 2008, Indianapolis, IN, USA* (2008), pp. 151–158.

[18] Capua, R.; Frota, Y.; Ochi, L. S.; Vidal, T. A study on exponential-size neighborhoods for the bin packing problem with conflicts. *Journal of Heuristics 24*, 4 (Aug 2018), 667–695.

[19] Chaves, A. A.; de Assis Correa, F.; Lorena, L. A. N. Clustering search heuristic for the capacitated p-median problem. In *Innovations in Hybrid Intelligent Systems* (Berlin, Heidelberg, 2007), E. Corchado, J. M. Corchado, and A. Abraham, Eds., Springer Berlin Heidelberg, pp. 136–143.

[20] Chaves, A. A.; Lorena, L. A. N. Clustering search algorithm for the capacitated centered clustering problem. *Computers & Operations Research 37*, 3 (2010), 552 – 558.

[21] Chaves, A. A.; Lorena, L. A. N. Hybrid evolutionary algorithm for the Capacitated Centered Clustering Problem. *Expert Systems with Applications 38*, 5 (2011), 5013–5018.

[22] Christensen, J.; Marks, J.; Shieber, S. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics 14* (1995), 203–232.

[23] Cook, W. J.; Cunningham, W. H.; Pulleyblank, W. R.; Schrijver, A. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[24] Cravo, G. L.; Ribeiro, G. M.; Lorena, L. A. N. A greedy randomized adaptive search procedure for the point-feature cartographic label placement. *Computers & Geosciences 34*, 4 (2008), 373 – 386.

[25] de Oliveira, D.; Baião, F. A.; Mattoso, M. Towards a taxonomy for cloud computing from an e-science perspective. In *Cloud Computing: Principles, Systems and Applications*, N. Antonopoulos and L. Gillam, Eds. Springer London, London, 2010, pp. 47–62.

[26] de Oliveira, D.; Ogasawara, E.; Baião, F.; Mattoso, M. Scicumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. In *3rd International Conference on Cloud Computing* (2010), pp. 378–385.

[27] Deelman, E.; Vahi, K.; Juve, G.; Rynge, M.; Callaghan, S.; Maechling, P. J.; Mayani, R.; Chen, W.; da Silva, R. F.; Livny, M., et al. Pegasus, a workflow management system for science automation. *FGCS 46* (2015), 17–35.

[28] Ennajjar, I.; Tabii, Y.; Benkaddour, A. Securing data in cloud computing by classification. In *Proceedings of the 2Nd International Conference on Big Data, Cloud and Applications* (New York, NY, USA, 2017), BDCA'17, ACM, pp. 49:1–49:5.

[29] Erickson, B. M. S.; Singh, R.; Evrard, A. E.; Becker, M. R.; Busha, M. T.; Kravtsov, A. V.; Marru, S.; Pierce, M.; Wechsler, R. H. A high throughput workflow environment for cosmological simulations. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the Campus and Beyond* (New York, NY, USA, 2012), XSEDE '12, ACM, pp. 34:1–34:8.

[30] Feo, T. A.; Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6* (1995), 109–133.

[31] Festa, P.; Resende, M. G. C. An annotated bibliography of GRASP part I: Algorithms. *International Transactions in Operational Research 16* (2009), 1–24.

[32] Festa, P.; Resende, M. G. C. An annotated bibliography of GRASP part II: Applications. *International Transactions in Operational Research 16* (2009), 131–172.

[33] Fleurent, C.; Glover, F. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing 11* (1999), 198–204.

[34] Forgy, E. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics 21*, 3 (1965), 768–769.

[35] Freire, J.; Koop, D.; Santos, E.; Silva, C. T. Provenance for Computational Tasks: A Survey. *Computing in Science and Engg. 10*, 3 (May 2008), 11–21.

[36] Gendreau, M.; Laporte, G.; Semet, F. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research 31* (2004), 347–358.

[37] Gendreau, M.; Potvin, J.-Y. *Handbook of Metaheuristics*, 2*nd* ed., vol. 146 of *International Series in Operations Research & Management Science*. Springer, 2010.

[38] Glover, F. Heuristics for integer programming using surrogate constraints. *Decision Sciences 8* (1977), 156–166.

[39] Glover, F. Tabu search and adaptive memory programing advances, applications and challenges. In *Interfaces in Computer Science and Operations Research* (1996), Kluwer, pp. 1–75.

[40] Glover, F.; Laguna, M.; Martí, R. Fundamentals of scatter search and path relinking. *Control and Cybernetics 39* (2000), 653–684.

[41] Goethals, B.; Zaki, M. J. Advances in frequent itemset mining implementations: report on fimi'03. *SIGKDD Exploration Newsletter 6* (2004), 109–117.

[42] Grahne, G.; Zhu, J. Efficiently using prefix-trees in mining frequent itemsets. In *Frequent Itemset Mining Implementations (FIMI'03)*, B. Goethals and M. J. Zaki, Eds. Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, 2003. Available at http://CEUR-WS.org/Vol-90.

[43] Grahne, G.; Zhu, J. Efficiently using prefix-trees in mining frequent itemsets, 2003.

[44] Guerine, M.; Rosseti, I.; Plastino, A. Extending the hybridization of metaheuristics with data mining: Dealing with sequences. *Intelligent Data Analysis 5* (2016), 1–24.

[45] Hafeez, I.; Ding, A. Y.; Suomalainen, L.; Hätönen, S.; Niemi, V.; Tarkoma, S. Demo: Cloud-based security as a service for smart iot environments. In *Proceedings of the 2015 Workshop on Wireless of the Students, by the Students for the Students* (New York, NY, USA, 2015), S3 '15, ACM, pp. 20–20.

[46] Hamming, R. W. Error detecting and error correcting codes. *Bell System Technical Journal 29*, 2 (1950), 147–160.

[47] Han, J.; Kamber, M. *Data Mining: Concepts and Techniques*, 3*rd* ed. Morgan Kaufmann Publishers, 2011.

[48] Han, J.; Pei, J.; Yin, Y. Mining frequent patterns without candidate generation. *SIGMOD Record 29* (2000), 1–12.

[49] Hey, T.; Tansley, S.; Tolle, K. M., et al. *The fourth paradigm: data-intensive scientific discovery*, vol. 1. Microsoft research Redmond, WA, 2009.

[50] Hoffa, C.; Mehta, G.; Freeman, T.; Deelman, E.; Keahey, K.; Berriman, B.; Good, J. On the use of cloud computing for scientific workflows. In *eScience* (2008), pp. 640–645.

[51] Imhof, E. Die anordnung der namen in der karte. *International Yearbook of Cartography 2* (1962), 93–129.

[52] Imhof, E. Positioning names on maps. *The American Cartographer 2*, 2 (1975), 128–144.

[53] Jackson, K. R.; Ramakrishnan, L.; Runge, K. J.; Thomas, R. C. Seeking supernovae in the clouds: A performance study. In *HPDC 2010* (New York, NY, USA, 2010), ACM, pp. 421–429.

[54] Jansen, K. An approximation scheme for bin packing with conflicts. *Journal of Combinatorial Optimization 3* (1999), 363–377.

[55] John M. Mulvey, H. P. C. Cluster analysis: An application of lagrangian relaxation. *Management Science 25*, 4 (1979), 329–340.

[56] Kaufman, L.; Rousseeuw, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*, vol. 33. Wiley, 2005.

[57] Kellerer, H.; Pferschy, U.; Pisinger, D. Introduction to np-completeness of knapsack problems. In *Knapsack problems*. Springer, 2004, pp. 483–493.

[58] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[59] Laguna, M.; Martí, R. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing 11* (1999), 44–52.

[60] Lin, S.; Kernighan, B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research 21* (1973), 498–516.

[61] Livny, J. Bioinformatic discovery of bacterial regulatory rnas using sipht. In *Bacterial Regulatory RNA: Methods and Protocols*, K. C. Keiler, Ed. Humana Press, Totowa, NJ, 2012, pp. 3–14.

[62] Lodi, A.; Allemand, K.; Liebling, T. M. An evolutionary heuristic for quadratic 0–1 programming. *European Journal of Operational Research 119* (1999), 662–670.

[63] Maia, M. R. d. H. Heurísticas híbridas com mineração de dados para o problema de roteamento de veículos com frota heterogênea. Master's thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2015.

[64] Maia, M. R. d. H.; Plastino, A.; Penna, P. H. V. Hybrid data mining heuristics for the heterogeneous fleet vehicle routing problem. *RAIRO-Oper. Res. 52*, 3 (2018), 661–690.

[65] Malluhi, Q.; Khan, K. M. Cloud computing without seeing. In *Proceedings of the First International Workshop on Security and Privacy Preserving in e-Societies* (New York, NY, USA, 2011), SeceS '11, ACM, pp. 42–44.

[66] Marks, J.; Shieber, S. The computational complexity of cartographic label placement, 1991. Technical report. Harvard University, USA.

[67] Martins, D.; Vianna, G. M.; Rosseti, I.; Martins, S. L.; Plastino, A. Making a state-of-the-art heuristic faster with data mining. *Annals of Operations Research 263*, 1 (Apr 2018), 141–162.

[68] Martins, S.; Rosseti, I.; Plastino, A. *Data Mining in Stochastic Local Search*. Springer International Publishing, Cham, 2018, pp. 39–87.

[69] Mattoso, M.; Werner, C.; Travassos, G. H.; Braganholo, V.; Ogasawara, E.; Oliveira, D.; et al. Towards supporting the life cycle of large scale scientific experiments. *IJBPIM 5*, 1 (2010), 79+.

[70] Mauri, G. R.; Ribeiro, G. M.; Lorena, L. A. N. A new mathematical model and a lagrangean decomposition for the point-feature cartographic label placement problem. *Computers & Operations Research 37* (2010), 2164–2172.

[71] Mladenović, N.; Hansen, P. Variable neighborhood search. *Computers & Operations Research 24* (1997), 1097–1100.

[72] Mulvey, J. M.; Beck, M. P. Solving capacitated clustering problems. *European Journal of Operational Research 18*, 3 (1984), 339–348.

[73] Murta, L.; Braganholo, V.; Chirigati, F.; Koop, D.; Freire, J. noworkflow: Capturing and analyzing provenance of scripts. In *Provenance and Annotation of Data and Processes - 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany. Revised Selected Papers* (2014), pp. 71–83.

[74] N, S.; R, B.; M, P. Cancer diagnosis epigenomics scientific workflow scheduling in the cloud computing environment using an improved pso algorithm. *Asian Pacific Journal of Cancer Prevention 19*, 1 (2018), 243–246.

[75] Negreiros, M.; Palhano, A. The capacitated centred clustering problem. *Computers & Operations Research 33*, 6 (2006), 1639–1663.

[76] Ocaña, K. A. C. S.; de Oliveira, D.; Ogasawara, E.; Dávila, A. M. R.; Lima, A. A. B.; Mattoso, M. SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes. In *Advances in Bioinformatics and Computational Biology* (Berlin, Heidelberg, 2011), O. Norberto de Souza, G. P. Telles, and M. Palakal, Eds., Springer Berlin Heidelberg, pp. 66–70.

[77] Ogasawara, E.; Dias, J.; Silva, V.; Chirigati, F.; Oliveira, D.; Porto, F.; Valduriez, P.; Mattoso, M. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation 25*, 16 (2013), 2327–2341.

[78] Oliveira, A.; Chaves, A. A.; Lorena, L. A. N. Clustering Search. *Pesquisa Operacional 33* (2013), 105–121.

[79] Oliveira, A.; Lorena, L. Hybrid evolutionary algorithms and clustering search. In *Hybrid Evolutionary Algorithms*, A. Abraham, C. Grosan, and H. Ishibuchi, Eds., vol. 75 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2007, pp. 77–99.

[80] Oliveira, C.; Urrutia, S. A.; Noronha, T. F. Heurística ILS para o problema da rotulação cartográfica de pontos. In *XII SPOLM – operational research and logistics brazilian navy symposium* (Rio de Janeiro, Brasil, 2009), vol. 84.

[81] Pereira, M.; Senne, E. A column generation method for the capacitated centred clustering problem. *VI ALIO/EURO workshop on applied combinatorial optimization (pp.* (2008), 1–6.

[82] Plastino, A.; Barbalho, H.; Santos, L. F. M.; Fuchshuber, R.; Martins, S. L. Adaptive and multi-mining versions of the DM-GRASP hybrid metaheuristic. *Journal of Heuristics 20* (2014), 39–74.

[83] Plastino, A.; Fuchshuber, R.; Martins, S. L.; Freitas, A. A.; Salhi, S. A hybrid data mining metaheuristic for the p-median problem. *Statistical Analysis and Data Mining 4* (2011), 313–335.

[84] Pradal, C.; Fournier, C.; Valduriez, P.; Cohen-Boulakia, S. Openalea: Scientific workflows combining data analysis and simulation. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management* (New York, NY, USA, 2015), SSDBM '15, ACM, pp. 11:1–11:6.

[85] Rabello, R. L.; Mauri, G. R.; Ribeiro, G. M.; Lorena, L. A. N. A clustering search metaheuristic for the point-feature cartographic label placement problem. *European Journal of Operational Research 234* (2014), 802–808.

[86] Resende, M. G. C.; Ribeiro, C. C. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*, 1st ed. Springer Publishing Company, Incorporated, 2016.

[87] Ribeiro, C. C.; Rosseti, I. `tttplots-compare`: A perl program to compare time-to-target plots or general runtime distributions of randomized algorithms. *Optimization Letters 9* (2015), 601–614.

[88] Ribeiro, G.; Mauri, G.; Lorena, L. A lagrangean decomposition for the maximum independent set problem applied to map labeling. *Operational Research 11*, 3 (2011), 229–243.

[89] Ribeiro, G. M.; Lorena, L. A. N. Heuristics for cartographic label placement problems. *Computers & Geosciences 32* (2006), 229–243.

[90] Ribeiro, G. M.; Lorena, L. A. N. Column generation approach for the point-feature cartographic label placement problem. *Journal of Combinatorial Optimization 15* (2008a), 147–164.

[91] Ribeiro, G. M.; Lorena, L. A. N. Lagrangean relaxation with clusters for point-feature cartographic label placement problems. *Computers & Operations Research 35* (2008b), 2129–2140.

[92] Ribeiro, M. H.; Plastino, A.; Martins, S. L. Hybridization of GRASP metaheuristic with data mining techniques. *Journal of Mathematical Modelling Algorithms 5* (2006), 23–41.

[93] Ribeiro, M. H.; Trindade, V. F.; Plastino, A.; Martins, S. L. Hybridization of GRASP metaheuristics with data mining techniques. In *Proceedings of the ECAI workshop on hybrid metaheuristics* (2004), pp. 69–78.

[94] Rochat, Y.; Taillard, R. D. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics 1* (1995), 147–167.

[95] Rynge, M.; Juve, G.; Kinney, J.; Good, J.; Berriman, G.; Merrihew, A.; Deelman, E. Producing an infrared multiwavelength galactic plane atlas using montage, pegasus and amazon web services. In *23rd Annual Astronomical Data Analysis Software and Systems, ADASS, Conference* (2013).

[96] Sakellariou, R.; Zhao, H.; Deelman, E. Mapping workflows on grid resources: experiments with the montage workflow. In *Grids, P2P and Services Computing*. Springer, 2010, pp. 119–132.

[97] Samarati, P.; di Vimercati, S. D. C. Data protection in outsourcing scenarios: Issues and directions. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (New York, NY, USA, 2010), ASIACCS '10, ACM, pp. 1–14.

[98] Santos, H. G.; Ochi, L. S.; Marinho, E. H.; Drummond, L. M. A. Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem. *Neurocomputing 70* (2006), 70–77.

[99] Santos, L. F.; Martins, S. L.; Plastino, A. Applications of the DM-GRASP heuristic: a survey. *International Transactions in Operational Research 15* (2008), 387–416.

[100] Scott, L. A.; Zimmerman, R.; Chang, H.-Y.; Heitzman, M.; Krajcik, J.; McNeill, K. L.; Quintana, C.; Soloway, E. Chemation: A handheld chemistry modeling and animation tool. In *Proceedings of the 2004 Conference on Interaction Design and Children: Building a Community* (New York, NY, USA, 2004), IDC '04, ACM, pp. 145–146.

[101] Siegel, S.; Castellan Jr, N. J. *Nonparametric Statistics for the Behavioral Sciences*, 2*nd* ed. McGraw-Hill, 1988.

[102] Sousa, V. S.; Leite, J.; Camata, J. J.; de Oliveira, D.; Coutinho, A. L. G. A.; Valduriez, P.; Mattoso, M. Raw data queries during data-intensive parallel workflow execution. *Future Generation Comp. Syst. 75* (2017), 402–422.

[103] Taylor, I.; Shields, M.; Wang, I.; Harrison, A. The triana workflow environment: Architecture and applications. In *Workflows for e-Science*. Springer, 2007, pp. 320–339.

[104] Taylor, I. J.; Deelman, E.; Gannon, D. B.; Shields, M. *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2014.

[105] Vaquero, L. M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A break in the clouds: Towards a cloud definition. *SIGCOMM Rev. 39*, 1 (Dec. 2008), 50–55.

[106] Verner, O. V.; Wainwright, R. L.; Schoenefeld, D. A. Placing text labels on maps and diagrams using genetic algorithms with masking. *INFORMS Journal on Computing 9* (1997), 266–275.

[107] Woodman, S.; Hiden, H.; Watson, P. Applications of provenance in performance prediction and data storage optimisation. *Future Generation Computer Systems 75* (2017), 299 – 309.

[108] Wozniak, J. M.; Armstrong, T. G.; Wilde, M.; Katz, D. S.; Lusk, E.; Foster, I. T. Swift/t: Scalable data flow programming for many-task applications. *SIGPLAN Not. 48*, 8 (Feb. 2013), 309–310.

[109] Yamamoto, M.; Camara, G.; Lorena, L. A. N. Tabu search heuristic for point-feature cartographic label placement. *Geoinformatica 6* (2002), 77–90.

[110] Zoraster, S. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research 5* (1990), 752–759.