

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

CARINE RODRIGUES DA COSTA

**Condução de Experimentos
Computacionais com Métodos
Heurísticos**

Goiânia
2011

CARINE RODRIGUES DA COSTA

Condução de Experimentos Computacionais com Métodos Heurísticos

Dissertação apresentada ao Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Otimização.

Orientador: Prof. Dr. Humberto José Longo

Goiânia
2011

CARINE RODRIGUES DA COSTA

Condução de Experimentos Computacionais com Métodos Heurísticos

Dissertação defendida no Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 30 de Março de 2011, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Humberto José Longo
Instituto de Informática – UFG
Presidente da Banca

Prof. Dra. Telma Woerle de Lima Soares
Instituto de Informática – UFG

Prof. Dr. Cláudio Nogueira de Meneses
Centro de Matemática, Computação e Cognição – UFABC

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador(a).

Carine Rodrigues da Costa

Graduou-se em Licenciatura Plena em Informática pela Universidade Federal de Mato Grosso em 2007. Durante sua graduação, foi pesquisadora do CNPq em um trabalho de iniciação científica no área de Física. Foi professora substituta nesta mesma universidade no período de 2007 a 2008, na área de Estrutura de Dados. Em 2009, ingressou no mestrado em Ciência da Computação na Universidade Federal de Goiás. Durante o mestrado foi monitora REUNI da disciplina de Linguagens Formais e Autômatos.

À minha querida e amada mãe, Vera Lúcia.

Agradecimentos

Durante o mestrado, na história que construí fizeram parte dela inúmeras pessoas, a qual devo também a conclusão deste trabalho. Primeiro, quero agradecer à Deus, e todas as vitórias que têm concedido em minha vida. Sem Ele na minha vida, nada seria possível.

Agradeço à minha mãe, Vera Lúcia, que me ensinou durante a vida inteira, que cuidou de mim como se eu fosse um tesouro, e que me apoiou em todos os momentos, mesmo estando longe, o amor nos unia. Sem ela eu não teria conquistado tudo o que consegui, ela é a minha força maior.

Agradeço à minha vó, Cecília, que me ajudou a dar o primeiro passo em direção ao mestrado. Me ajudou e apoiou até o final.

Agradeço ao Thiago, meu eterno companheiro, que viveu ao meu lado todos os momentos do mestrado, sendo meu companheiro, amigo, dando conselhos, correções no meu texto, apoio, abraços que eu tanto precisava para seguir em frente. Também agradeço à sua família, que sempre me receberam com amor e carinho, me tornando parte dela.

Agradeço às minhas irmãs, Morgani e Brenda, companheiras que me proporcionaram horas de conversa, quando eu mais precisava. Sem dúvida fazem parte desta história.

Agradeço ao meu pai, Paulo, e toda minha família, que compreenderam minha ausência.

Agradeço ao meu orientador, Professor Humberto José Longo, por me receber e me aceitar, e pelas inúmeras correções em meu texto.

Agradeço ao Professor Cláudio Nogueira de Meneses, pelo apoio e incentivo inestimáveis.

Agradeço à todos os professores que também contribuíram para a realização deste sonho, com as disciplinas, conselhos, apoio e amizades.

Agradeço aos queridos técnicos que sempre me atenderam com muito carinho e dedicação, com empenho em seus trabalhos, em especial Edir Borges, Enio Perez e Ricardo Sena.

Por fim, agradeço aos meus queridos amigos, que me compreendiam ou simplesmente ouviam as minhas dificuldades e vitórias. Tanto eu pedia ajuda, e quando pedia, sempre tinha alguém para me confortar. Bete Kowata, Valdemar Neto, Patrícia Fernandes,

Fabiana Freitas, Bruno Calçado, Elisângela Dias, Adriana Rocha, Luiz Loja, Jean Martins, Luciana Nishi, Renan Rodrigues, Jair Alarcon, Enio Perez, Bruno Machado, Santiago Valdes, foram meu ombro amigo, me propiciaram longas conversas e compartilharam dos mesmos momentos no mestrado. Aos amigos Johnys Cavalcante, Dalva Nunes, Angélica Oliveira, Nádia Cotrim e Cristiane Silva, que mesmo distantes, estavam comigo.

“Toda a teoria deve ser feita para poder ser posta em prática, e toda a prática deve obedecer a uma teoria.”

Fernando Pessoa,
Palavras iniciais da Revista de Comércio e Contabilidade.

Resumo

Costa, Carine Rodrigues da. **Condução de Experimentos Computacionais com Métodos Heurísticos**. Goiânia, 2011. 149p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

A necessidade de resolver problemas de otimização em um limite razoável de tempo computacional faz com que o desenvolvimento de heurísticas seja uma grande área de pesquisa. Usualmente, heurísticas desenvolvidas para problemas de otimização são avaliadas empiricamente, pela sua aplicação a um conjunto de instâncias específicas, comparando qualidade da solução e esforços computacionais. Além disso, ao se apresentar uma nova heurística, as contribuições devem ser avaliadas cientificamente e relatadas de uma maneira objetiva. Ao descrever um experimento computacional e relatar os resultados obtidos do mesmo, pode ficar evidente a dificuldade de reproduzir o experimento ou comparar os resultados obtidos com os de outros experimentos. Parte da origem dessas questões vem do fato de que não há padrão para o relato de experimentos na área de Computação. Portanto, o foco deste trabalho é investigar métodos de condução de pesquisa experimental com heurísticas, para analisar quais são os mais favoráveis e consistentes na avaliação destas. Desta forma, a investigação resultou em uma compilação com a contribuição de diversos autores, em que consistiu na identificação de um conjunto de recomendações, com a elaboração de um *checklist*, representando de forma sumarizada todos os itens vistos nesta pesquisa. Os resultados dessa revisão serviram como base para a definição da pesquisa e condução de um estudo exemplo, que consistiu na análise de artigos que tratam do Problema de Atribuição Quadrática (PAQ), com a verificação dos itens necessários para compreensão, reprodução e comparação dos experimentos realizados.

Palavras-chave

Otimização, Métodos Heurísticos, Metaheurísticas, Condução de Experimentos Computacionais

Abstract

Costa, Carine Rodrigues da. **Conduction of Computational Experiments with Heuristic Methods**. Goiânia, 2011. 149p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The necessity of solving optimization problems in a reasonable computational time limit makes the development of heuristics be a large research area. Usually, developed heuristics for optimization problems are empirically evaluated by its application to a set of specific instances, comparing to quality solution and computational efforts. Besides, when presenting a new heuristic, the contributions should be scientifically evaluated and reported in an objective way. The quality of a computational experiment report may become evident the difficulty to reproduce the experiment or compare the results with those of other experiments. Part of the origin of these issues comes from the fact that there is no standard for reporting experiments in Computer Science. Therefore, the focus of this work is to investigate methods of conducting experimental research with heuristics, to examine what methods are more favorable and consistent in evaluating these. Thus, the investigation resulted in a compilation with contribution of several authors, which consisted in identifying a set of recommendations, including the formulation of a checklist representing the summary form of all the items that were seen in this study. The results of this review served as the basis for definitining the research and leading a sample study, which consisted in analysis of articles that deal with the Quadratic Assignment Problem (QAP), by checking the necessary items for understanding, reproduction and comparison of the performed experiments.

Keywords

Optimization, Heuristic Methods, Metaheuristics, Conduction of Computational Experiments

Sumário

Lista de Figuras	12
Lista de Tabelas	13
1 Introdução	14
1.1 Organização do Trabalho	17
2 Métodos Heurísticos	19
2.1 Definições Preliminares	20
2.2 Métodos Heurísticos	23
2.2.1 Busca Local	24
2.3 Métodos Metaheurísticos	27
2.3.1 <i>Variable Neighborhood Search</i>	29
2.3.2 <i>Simulated Annealing</i>	31
2.3.3 Busca Tabu	32
2.3.4 GRASP	35
2.3.5 Algoritmos Genéticos	37
2.3.6 Colônia de Formigas	41
2.3.7 <i>Path Relinking</i>	45
2.3.8 Times Assíncronos	46
3 Condução de Experimentos com Heurísticas	51
3.1 Análise de Algoritmos	51
3.1.1 Análise Assintótica	52
3.1.2 Experimentação	53
3.2 Passos para Condução de Experimentos Utilizando Heurísticas	55
3.3 Revisão da Literatura	55
3.4 Objetivos do Experimento	56
3.5 Medidas de Desempenho e Fatores a Explorar	58
4 Projeto e Execução do Experimento	64
4.1 Planejamento Experimental	64
4.2 Seleção do Conjunto de Instâncias de Teste	68
4.2.1 Conjunto de Instâncias de Teste Reais	69
4.2.2 Variações nos Conjuntos de Instâncias de Teste Reais	69
4.2.3 Bibliotecas Públicas de Referência	69
4.2.4 Instâncias Geradas Aleatoriamente	70
4.2.5 Como Gerar um Conjunto de Instâncias de Teste	71
4.3 Critérios de Parada	73

4.4	Execução do Experimento	74
4.5	Ajustes de Parâmetros	75
4.6	Questões de Implementação	76
4.7	Tempo Gasto na Execução do Experimento	76
5	Análise de Dados e Relato do Experimento	79
5.1	Análise dos Dados	79
5.1.1	Estimativas Estatística de Valores Ótimos	83
5.2	Relato dos Resultados dos Experimentos	85
5.2.1	Apresentação dos Resultados	86
5.2.2	Relatando a variância	88
5.2.3	Reprodução e Comparação do Experimento	89
5.2.4	Falhas ao Relatar os Resultados	93
5.3	<i>Checklist</i> para Avaliação de Relato de Experimento Computacional	94
6	Estudo Exemplo: Problema de Atribuição Quadrática	98
6.1	Problema de Atribuição Quadrática	99
6.2	Artigos Seleccionados	102
6.3	Análise dos Artigos	104
6.3.1	Revisão da Literatura	104
6.3.2	Modelo Experimental	106
6.3.3	Apresentação dos Algoritmos	109
6.3.4	Implementação	111
6.3.5	Relato dos Resultados	113
6.3.6	Conclusões	117
6.4	Conclusões do Capítulo	118
7	Considerações Finais	120
	Referências Bibliográficas	123
A	Conceitos Básicos de Estatística	132
A.1	Planejamento Experimental	132
A.2	Princípios básicos de um planejamento de experimentos	134
A.3	Cálculos Estatísticos Básicos	136
A.4	Métodos Estatísticos para Análise	141
A.4.1	Análise de Variância de um Fator	141
A.4.2	Fatorial Completo	144
A.4.3	Quadrado Latino	146
B	Itens Para avaliação de Experimentos	148

Lista de Figuras

2.1	Classes de problemas consideradas neste trabalho. Retirada de [85].	21
2.2	Exemplos de possíveis movimentos sobre a vizinhança de s .	25
2.3	Exemplo de movimento k -opt para o Problema do Caixeiro Viajante, com $k = 2$.	27
2.4	Elementos de um Algoritmo Genético.	38
2.5	Procedimento Básico de um Algoritmo Genético.	39
2.6	Procedimento Básico de <i>crossover</i> .	41
2.7	Exemplo de A-Team 1 composição, 3 memórias básicas e 6 agentes. Retirado de [5].	47
4.1	Abordagem básica: instâncias $\times$ algoritmos.	65
A.1	Um sistema representado por uma função ligando os fatores (variáveis de entrada) às respostas (variáveis de saída). Retirado de [10].	145
A.2	Exemplo de quadrado latino de tamanho 4	146

Lista de Tabelas

5.1	Medidas de variância em uma tabela. Baseado em [95].	89
5.2	Checklist para Relato de Experimento Computacional Proposto.	97
6.1	Artigos selecionados para análise.	103
6.2	Itens cobertos sobre a Revisão da Literatura	105
6.3	Itens cobertos sobre o Modelo Experimental.	109
6.4	Itens cobertos na Apresentação dos Algoritmos.	111
6.5	Itens cobertos sobre a Implementação dos Algoritmos.	113
6.6	Itens cobertos sobre o Relato dos Resultados.	116
6.7	Itens cobertos sobre as Conclusões.	118
A.1	Dados típicos de um Experimento com um Fator.	141
B.1	Checklist de relato de experimento computacional, retirado de Crowder, Dembo e Mulvey [19].	149

Introdução

Quando é desenvolvida uma pesquisa, esta deve ser descrita em detalhes, para que seja possível fazer uma análise sobre semelhanças e diferenças em relação ao que já foi estudado. Além disso, deve permitir a comparação ou reprodução de determinadas situações, para que então seja verificado se os resultados encontrados têm total ou parcial similaridade. Para isto é necessário compreender em que contexto uma pesquisa se enquadra, de acordo com os objetivos da mesma.

A condução de uma pesquisa, de acordo com Rardin e Uzsoy [95], pode ser guiada de acordo com três princípios: **pesquisa versus desenvolvimento; projeto, planejamento e aplicações de controle** e **ciclo de vida** do problema em estudo. Estes podem ser aplicados em diversas pesquisas na área de computação. Entretanto, nesta dissertação os mesmos serão descritos em relação às investigações experimentais com heurísticas.

Em relação ao contexto **pesquisa × desenvolvimento**, a **pesquisa** é destinada a descobrir novas técnicas e/ou tecnologias para problemas, ou aplicar uma técnica/tecnologia já existente de maneira criativa para novos problemas. O interessante é inovar. Já na fase de **desenvolvimento** são implementados algoritmos com o objetivo de encontrar soluções com a qualidade desejada. Os detalhes de implementação se tornam importantes, porque o foco é sobre como implementar o algoritmo de forma eficiente e como configurar seus parâmetros para alcançar os resultados desejados [76, 95].

O contexto de **projeto, planejamento e controle de aplicações** se refere ao tempo disponível para obter uma solução, dependendo do problema em questão. Os **problemas de projeto** cobrem um longo período de tempo, tais como projeto de redes de telecomunicações, etc. Os **problemas de controle** geralmente envolvem decisões sobre um horizonte de tempo pequeno. Um exemplo seria a transmissão de dados numa rede de computadores. Os **problemas de planejamento**, como Problemas de Programação de Horários (*Timetabling*), ocupam uma posição intermediária em termos de frequência e tempo de solução disponíveis. Em geral, para estes problemas, uma solução deve ser obtida rapidamente, o que torna o uso de métodos exatos impraticável.

Por último, o contexto de **ciclo de vida do problema**, diz respeito à evolução das modelagens matemáticas e dos algoritmos desenvolvidos. Quando um problema é

escolhido para ser estudado, deve ser levado em conta tudo o que já foi desenvolvido anteriormente, pois ao se estudar o ciclo de vida, pode-se chegar aos mesmos resultados, que já foram encontrados anteriormente. Mostrar que determinado algoritmo resolve um problema não é suficiente, pois deve-se situar o problema/algoritmo em relação ao ciclo de vida dele.

Quando o ciclo de vida de um problema é estudado, torna-se possível saber o que ainda falta estudar e o que pode ser melhorado. Desta forma pode-se produzir uma contribuição que não foi encontrada em abordagens anteriores, ou apresentar um algoritmo que supere métodos existentes ou alguma medida de desempenho relevante. A experimentação necessita ser bem explicada, mostrando que a heurística proposta resolve várias instâncias e pode ser comparada com outros algoritmos. Problemas clássicos como Problema do Caixeiro Viajante, Problema da Mochila, Escalonamento de Tarefas, que têm sido estudados por décadas, possuem um ciclo de vida bem definido, que deve ser compreendido, caso se queira estudar um tal problema.

Enfim, destes três grupos principais, este trabalho se enquadra no contexto de pesquisa $\times$ desenvolvimento, pois são investigados métodos de condução de pesquisa experimental com heurísticas, para analisar quais são os métodos mais favoráveis e consistentes na avaliação de heurísticas.

Heurísticas são utilizadas para a resolução de problemas de otimização. Um problema de otimização é composto de um conjunto de restrições e uma função objetivo, ambos associados a variáveis de decisão. A meta ao se resolver um tal problema é encontrar uma atribuição de valores às variáveis de decisão, segundo limites impostos pelo conjunto de restrições, que otimize o valor da função objetivo.

Os métodos heurísticos procuram boas soluções viáveis, não necessariamente uma solução ótima, em circunstâncias em que a complexidade do problema é excessiva ou o tempo disponível para sua resolução é limitado.

A necessidade de resolver problemas de otimização em um limite razoável de tempo faz com que o desenvolvimento de heurísticas seja uma grande área de pesquisa. Ao contrário dos algoritmos exatos, em que tempo e eficiência são as principais medidas de sucesso, existem pelo menos duas outras questões importantes na avaliação de heurísticas: a rapidez com que as soluções podem ser obtidas e quanto elas se aproximam de uma solução ótima [95].

Usualmente, heurísticas desenvolvidas para problemas de otimização são avaliadas empiricamente, através de sua aplicação a um conjunto de instâncias específicas, comparando a qualidade de soluções e esforços computacionais [95]. Além disso, ao se apresentar uma nova heurística, as contribuições devem ser avaliadas cientificamente e relatadas de uma maneira objetiva. Mas nem sempre isto é feito [6].

Vários pesquisadores têm buscado definir as diretrizes para a pesquisa empírica

em algoritmos (Crowder et al, 1979 [19]; Lin e Rardin, 1980 apud [95]; Golden e Stewart, 1985 apud [95]; Golden et al., 1986 apud [95]; Greenberg, 1990 [43]; Jackson et al., 1990 apud [95]; Lee et al., 1993 apud [95]; Barr et al., 1994 [6]; Hooker, 1994 [53] e 1995 [54]; Ahuja e Orlin, 1996 [2]; McGeogh, 1996 [67]), a maioria dirigida ao ensaio de métodos exatos. Embora muitas questões que surgem sejam idênticas às verificadas na avaliação de algoritmos exatos, a natureza da otimização heurística apresenta uma série de desafios. Em particular, deve-se muitas vezes avaliar se a qualidade de uma solução encontrada pode ser avaliada em relação a uma solução ótima obtida com métodos exatos, ou se há alguma estimativa para avaliar a solução [95].

Como um algoritmo é uma abstração, ele pode ser avaliado experimentalmente. Neste contexto, um experimento consiste em encontrar soluções para uma série de instâncias de um problema usando uma implementação de um algoritmo. O pesquisador deve selecionar as instâncias, escolher um ambiente computacional, escolher as medidas de desempenho, configurar os parâmetros do algoritmo, e finalmente relatar os resultados (geralmente o comportamento dos algoritmos). A escolha feita para cada um destes fatores pode ter um efeito substancial sobre os resultados e a relevância do experimento. Algumas questões em relação ao desempenho do algoritmo ou em relação à classe de instâncias a se trabalhar são fáceis de responder, até mesmo antes de se implementar um algoritmo, mas outras só poderão ser respondidas com a execução dos experimentos.

Ao descrever um experimento computacional e relatar os resultados obtidos do mesmo, pode ficar evidente a dificuldade de reproduzir o experimento ou de comparar os resultados obtidos com os de outros experimentos. Portanto, com o objetivo de conduzir melhor um experimento, partindo do início, da definição dos objetivos, até o final, o relato dos resultados, surgem algumas questões, que delineiam o presente trabalho:

- Quais são os passos a serem seguidos na condução de experimentos computacionais com métodos heurísticos?
- O que realmente deve ser relatado em um experimento computacional?
- Quais são os requisitos mínimos para tornar um trabalho passível de comparação e reprodução?
- Qual o conjunto de critérios de qualidade que deve ser atendido ao relatar um experimento computacional utilizando métodos heurísticos?
- Qual melhoria é obtida na execução de um experimento computacional ao utilizar planejamento experimental?
- Em que um modelo experimental pode contribuir na condução de métodos heurísticos?

Parte da origem destas questões vem do fato que não há padrão para o relato de experimentos na área de computação [19]. Portanto, um dos principais objetivos deste

trabalho é fazer uma compilação de várias sugestões listadas por diversos pesquisadores. O trabalho fundamenta-se em tentar responder as questões que foram levantadas. Dessa forma, os objetivos específicos são:

- Fazer um levantamento dos métodos empregados por diferentes pesquisadores, para o relato de experimentos. Descrever vantagens e desvantagens em relação aos experimentos computacionais utilizando métodos heurísticos e dar algumas sugestões para que pesquisadores possam escrever e avaliar melhor seus relatórios de experimentos computacionais.
- Aplicar as sugestões encontradas na literatura, fazendo um estudo exemplo que consistirá na análise de relato de artigos sobre o Problema de Atribuição Quadrática (PAQ), verificando se o relato segue as recomendações encontradas no levantamento feito.
- Reunir em um único documento várias sugestões listadas por diversos pesquisadores.

1.1 Organização do Trabalho

Inicialmente, o Capítulo 2 define vários conceitos que são utilizados na área de Otimização, e alguns dos métodos heurísticos mais utilizados. Foi feita uma revisão bibliográfica para identificação das recomendações e passos a serem seguidos na realização de experimentos com algoritmos heurísticos. Em resumo, esta revisão resultou em uma compilação com a contribuição de diversos autores, tais como Barr et al. [6], Crowder et al. [19], Johnson [57], McGeoch [67], Moret [76], Rardin e Uzsoy [95], entre outros, e foram selecionados como os principais passos a serem seguidos para realizar experimentos com algoritmos:

1. Fazer uma revisão da literatura;
2. Definir os objetivos do experimento;
3. Escolher medidas de desempenho e fatores a explorar;
4. Projetar e executar o experimento;
5. Analisar os dados e mostrar as conclusões; e
6. Relatar os resultados dos experimentos.

No passo 2, o objetivo da pesquisa deve ser especificado claramente, deve ser o ponto de partida. A partir dele é que serão respondidas as questões, na qual a experimentação é necessária. É nesta fase que são listadas as hipóteses a serem testadas, os resultados a procurar e quais fatores explorar. Nesta fase também é definida uma classe de problemas a trabalhar.

Definidos os objetivos, no passo 3 serão escolhidas as medidas de desempenho e fatores a serem explorados. As medidas de desempenho podem ser divididas em três tipos: qualidade da solução, esforço computacional e robustez. Em relação à qualidade das soluções, busca-se saber como a heurística trata a otimalidade. Em relação aos métodos heurísticos, a velocidade de computação é um fator chave. Em relação à robustez, uma heurística que encontra soluções de qualidade para poucas instâncias de um problema não é robusta e também não é interessante [6]. Os passos 1, 2 e 3 são descritos no Capítulo 3.

O passo 4, projetar e executar o experimento, consiste em escolher um modelo experimental que se adeque ao problema em questão. Um bom experimento deve alcançar as metas experimentais, demonstrar claramente o desempenho dos testes, ter justificativas lógicas, gerar boas conclusões e ser passível de reprodução. Todas estas características têm um valor importante nos testes dos métodos heurísticos. Além de escolher um modelo experimental, também são feitas nesta fase a seleção ou geração do conjunto de instâncias de teste, são executados os testes e feitos ajustes nos parâmetros. Este passo é explicado no Capítulo 4.

O passo 5, a análise de dados, consiste em converter os dados coletados em informações através da análise e interpretação. A análise de dados consiste em avaliar os dados que foram obtidos, aplicando técnicas estatísticas e não estatísticas com relação aos objetivos definidos no início do experimento.

O último passo, o relato dos experimentos, tem por objetivo mostrar as contribuições obtidas. Alguns itens importantes a serem relatados são: como os detalhes de implementação, configuração de parâmetros, heurísticas e escolhas de estruturas de dados afetaram o tempo de execução do algoritmo; quais são os gargalos computacionais na prática e como eles dependem do tamanho da instância; comparação do tempo de execução encontrado com o tempo dos principais concorrentes, dentre outros. Os passos 5 e 6 encontram-se detalhados no Capítulo 5. Ainda neste capítulo, é apresentado o *checklist* desenvolvido, em que foram listados e organizados todos os itens recomendados neste trabalho para a condução e relato de experimentos.

Os resultados desta revisão serviram como base para a definição da pesquisa e condução do estudo exemplo, que está contido no Capítulo 6. Foi feita uma análise sobre os relatos dos experimentos computacionais realizados por alguns artigos muito citados na literatura. Estes artigos tratam do Problema de Atribuição Quadrática (PAQ). A análise destes trabalhos consistiu na verificação dos itens necessários para compreensão, reprodução e comparação dos experimentos realizados.

Finalmente, o Capítulo 7 apresenta as conclusões e possíveis trabalhos futuros.

Como foram necessários alguns conceitos básicos de Estatística, esses foram descritos no Apêndice A. O Apêndice B contém o *checklist* proposto por Crowder, Dembo e Mulvey [19] para avaliação de relato de experimento computacional.

Métodos Heurísticos

Uma grande variedade de problemas de otimização pertence à classe de problemas NP-Difíceis, isto é, não se sabe se existem algoritmos de complexidade de tempo polinomial para solucioná-los. Desta maneira, torna-se impraticável solucionar muitos problemas de forma exata, já que em muitos casos o tempo disponível para solucioná-los é razoavelmente curto. Por isto, existem algoritmos que não garantem uma solução ótima, porém, em geral dão uma solução “suficientemente boa”. Dessa forma, existem três possibilidades para se resolver tais problemas na prática [77]:

1. **Algoritmos super-polinomiais:** Em alguns casos existem algoritmos que são super-polinomiais e executam razoavelmente rápido na prática. Por exemplo, o Problema da Mochila pertence à classe NP-Difícil mas é considerado fácil, já que existe um algoritmo “pseudo-polinomial” para este problema. Dentre as técnicas utilizadas destacam-se o *branch-and-bound* e programação dinâmica. Um problema dessa abordagem é que poucos problemas são susceptíveis a essas técnicas.
2. **Análise probabilística de heurísticas:** Outra possibilidade é deixar a exigência de que uma solução para um problema atenda igualmente todas as restrições do mesmo. Na análise probabilística são assumidas hipóteses sobre a distribuição probabilística das entradas do algoritmo, e são derivados resultados analíticos sobre a saída do algoritmo.
3. **Algoritmos de aproximação:** Para diversos problemas, é possível relaxar o requisito de sempre encontrar uma solução ótima. Parece razoável implementar algoritmos que são realmente eficientes para resolver problemas NP-difíceis, com um custo de prover soluções que em todos os casos é garantida uma solução subótima.

Em relação à essas três possibilidades, será abordada neste trabalho somente a segunda. O objetivo deste capítulo é oferecer uma revisão dos principais métodos heurísticos e metaheurísticos citados na literatura. Para isso, serão definidos algumas conceitos importantes relacionados à otimização descritos na Seção 2.1, definição de heurística e métodos na Seção 2.2, definição de metaheurística e métodos meta-heurísticos mais utilizados na Seção 2.3.

2.1 Definições Preliminares

Segundo Papadimitriou e Steiglitz [85], muitos problemas, tanto de interesse teórico quanto prático, preocupam-se com a escolha da “melhor” configuração ou conjunto de parâmetros para alcançar algum objetivo. Durante as últimas décadas surgiu uma hierarquia de problemas, juntamente com um conjunto de técnicas para resolução desses. De acordo com essa hierarquia, a classe que abrange a maior parte dos problemas é chamada de **problemas de programação não linear**, é definida da seguinte maneira:

Encontrar x tal que:

$$\begin{cases} \text{minimize } f(x), \\ \text{sujeito a} \\ \quad g_i(x) \geq 0, \text{ para } i = 1, \dots, m, \\ \quad h_j(x) = 0, \text{ para } j = 1, \dots, p, \end{cases}$$

onde f , g_i e h_j são funções de $\mathbb{R}^n$ em $\mathbb{R}$.

Quando f é uma função convexa, g_i é uma função côncava e h_j é uma função linear, o problema é chamado de **problema de programação convexa**. Quando f , g_i e h_j são lineares, o problema é chamado de **problema de programação linear**. Nessa última classe de problemas, esses são chamados de **combinatoriais**, pois possuem um conjunto de possíveis soluções definidas por um conjunto de vértices de um poliedro definido por restrições lineares.

A Figura 2.1 mostra de uma maneira simples como se relacionam as classes de problemas. A maior classe equivale aos problemas de programação não-linear. Já os problemas de programação convexa, programação inteira e linear são subconjuntos da classe de problemas de programação não-linear. A classe de problemas de programação linear é um subconjunto da classe de problemas convexas. Entre as classes de problemas de programação convexa e problemas de programação inteira estão alguns problemas como Fluxo em Redes e *Matching*.

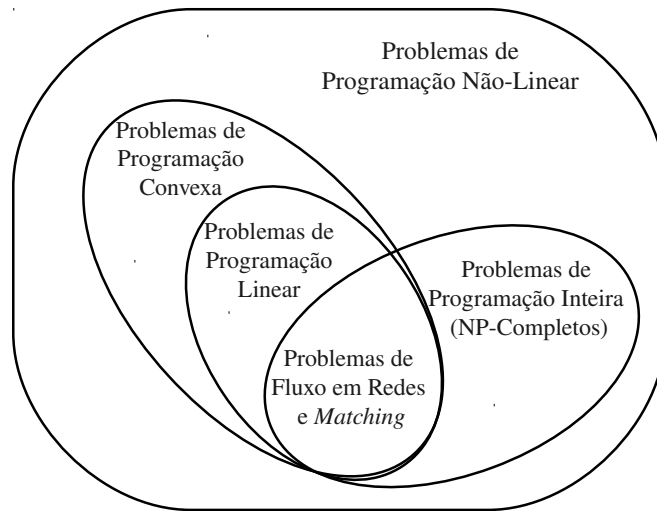


Figura 2.1: *Classes de problemas consideradas neste trabalho.*
Retirada de [85].

Problemas de otimização podem ser modelados por meio de um conjunto de variáveis com seus domínios e restrições relativas às definições das variáveis. Eles podem ser divididos em três categorias: os que têm exclusivamente **variáveis discretas**; os que têm exclusivamente **variáveis contínuas**; e os que possuem **variáveis contínuas e discretas**. Nos problemas que envolvem variáveis contínuas, geralmente procura-se um conjunto de números reais de uma dada função; já nos problemas que envolvem variáveis discretas procura-se por um objeto de um conjunto finito ou possivelmente infinito, e este objeto pode equivaler a um inteiro, um conjunto, uma permutação ou um grafo. As técnicas descritas neste capítulo, trabalham sobre o domínio das variáveis discretas, que resolvem problemas que pertencem à classe de problemas de otimização combinatória [4].

As definições desta seção estão baseadas em [4, 72, 85, 109]. A seguir, são apresentadas as definições básicas de otimização, necessárias para compreensão do trabalho. As definições estão considerando sempre um problema de minimização.

Definição 2.1 Um **problema de otimização** Π é especificado por um conjunto I de instâncias de um problema que pode ser um problema de minimização ou maximização.

Definição 2.2 Uma **instância** de um problema de otimização é um par (S, c) , onde S é o domínio de soluções factíveis; c é o custo de uma solução, um mapeamento que pode ser representado por $c : S \rightarrow \mathbb{R}$, ou seja, uma função de custo que associa a cada solução pertencente a S um valor real.

Definição 2.3 O **tamanho de uma instância** corresponde ao total de códigos (numéricos e alfanuméricos) necessários para sua identificação, considerando o tipo e a estrutura dos dados utilizados.

Definição 2.4 Uma solução $s \in S$ é uma solução ótima global (ótimo global) se $c(s) \leq c(s'), \forall s' \in S$. O ponto c é chamado de **solução ótima global** para uma dada instância, ou simplesmente **solução ótima**. Este tipo de solução será indicada por s^* .

Definição 2.5 Na resolução de um problema de otimização combinatória, o objetivo é encontrar uma solução ótima $s^* \in S$.

Em relação à algoritmos que tentam resolver problemas de otimização combinatória, uma característica importante é a função de vizinhança. Esta função especifica para cada solução, quais soluções estão mais próximas dela. A função de vizinhança é geralmente definida em termos de pequenas mudanças que podem ser aplicadas à solução para obter uma solução vizinha.

Definição 2.6 Uma **função de vizinhança** $N : S \rightarrow 2^S$, onde 2^S corresponde ao conjunto $\{V \mid V \subseteq S\}$. A vizinhança especifica para cada solução $s \in S$ um conjunto $N(s) \subseteq S$ chamado vizinhança de s . A cardinalidade de $N(s)$ é chamada de tamanho da vizinhança de s .

De maneira simples, a vizinhança pode ser entendida como um conjunto $N(s)$ de pontos que estão próximos aos pontos $s \in S$. Em vários problemas combinatoriais, a escolha de N pode depender criticamente da estrutura de S .

Definição 2.7 Cada solução $s' \in N(s)$ é chamada de **vizinha** de s .

Definição 2.8 Um **movimento** é uma modificação m que transforma uma solução s em outra, s' , que esteja em sua vizinhança. Representa-se esta operação por $s' \leftarrow s \oplus m$.

Definição 2.9 O **grafo de vizinhança** de uma instância (S, c) de uma problema de otimização combinatória, associado a uma função de vizinhança N , é um grafo direcionado $G = (S, A)$, onde S representa um conjunto de vértices que equivale ao conjunto S de soluções, e o conjunto de arcos A é definido de tal forma que $(s, s') \in A$ se e somente se $s' \in N(s)$. O peso de um nó é o custo da solução correspondente. Se a função de vizinhança for simétrica, então o grafo pode ser simplificado em um grafo não-direcionado, substituindo-se os arcos (s, s') e (s', s) pela aresta $\{s, s'\}$.

Encontrar uma solução ótima global de uma instância para alguns problemas pode ser difícil, mas geralmente é possível encontrar uma solução s' que é a melhor da vizinhança $N(s)$.

Definição 2.10 Uma solução s' é dita **alcançável** a partir de uma solução s se o grafo de vizinhança G contiver um caminho de s até s' .

Definição 2.11 Dada uma instância (S, c) de um problema de otimização e uma vizinhança N , uma solução viável $s \in S$ é chamada **ótimo local** em relação a N (ou simplesmente **ótimo local**) se $c(s) \leq c(s'), \forall s' \in N(s)$. Este tipo de solução será indicada por $\hat{s}$.

Pelo fato de geralmente ser interessante encontrar um ótimo global e porque muitos algoritmos podem calcular mais do que um ótimo local, é importante saber se um ótimo local é ou não global. Isto depende da vizinhança N . A definição a seguir ilustra a situação em que um ótimo local é também um ótimo global.

Definição 2.12 Dado um problema de otimização com um conjunto S viável e uma vizinhança N , se sempre que $s \in S$ é um ótimo local em relação a N é também um ótimo global, então a vizinhança N é **exata**.

Definição 2.13 Seja $\hat{s} \in S$ um ótimo local. A profundidade de $\hat{s}$ é definida como a distância mínima de um caminho p de $\hat{s}$ a uma solução s , com $c(s) < c(\hat{s})$. Se a solução s não existe, então a profundidade de $\hat{s}$ é ∞ .

Definição 2.14 Um **limite inferior** para um problema é um valor menor ou igual ao custo associado à função objetivo de uma solução ótima para o problema. Considerando o espaço de solução S , temos que: $li(S) \leq f(s^*), s^* \in S$.

Definição 2.15 Um **limite superior** para um problema é um valor maior ou igual ao custo associado à função objetivo de uma solução ótima para o problema. Considerando o espaço de solução S , temos que: $ls(S) \geq f(s^*), s^* \in S$.

2.2 Métodos Heurísticos

A palavra heurística tem origem da palavra grega *eurisco*, que significa “Eu descubro”. A origem desta palavra está relacionada com a explicação de Poyla, que define heurística como “o estudo dos métodos e das regras de descoberta e invenção”[91]. Portanto, heurística pode ser considerada como um desenvolvimento de métodos e regras baseadas em métodos não dedutivos, podendo ser entendida como um caso especial do método de tentativa e erro, no qual os problemas são solucionados através de tentativas, até encontrar uma solução viável para o problema.

De acordo com Rosa e Orey [97], heurística pode ser entendida como um método que não utiliza suposições arbitrárias, mas que aplica uma qualificada base de conceitos, modelos e hipóteses no processo de resolução de problemas. Por isso, a heurística difere do método dedutivo em relação a aplicação de suposições, analogias e hipóteses, pois utiliza diferentes tipos de modelos para solucionar problemas. A heurística pode ser

considerada como um movimento realizado para avançar uma estratégia particular de pesquisa. Contudo, a heurística é um processo iterativo e não há garantia de que a solução para um problema possa ser encontrada na primeira tentativa.

O conceito de heurística como uma alternativa para resolução de problemas de otimização foi introduzido por Glover, em 1986 [37]. Voss [110] define **heurística** como uma técnica, consistindo de uma regra ou um conjunto de regras, que busca boas soluções em um tempo computacional razoável. Uma heurística é chamada de aproximativa, pois encontra boas soluções com pouco esforço computacional, entretanto não garante otimalidade. A qualidade da solução pode ser definida por uma métrica de avaliação ou critério. A partir dessa especificação, as soluções encontradas podem ser tomadas como viáveis.

O propósito geral dos métodos heurísticos consiste em identificar soluções de um problema, onde o tempo é mais importante que a qualidade da solução, ou o conhecimento da qualidade. Muitos métodos heurísticos são associados com problemas em que existe uma solução ótima e esta pode ser computada por um algoritmo exato. Esses métodos são geralmente utilizados para identificar boas aproximações de soluções, em menos tempo que um algoritmo exato levaria para descobrir uma solução ótima. O uso de métodos exatos pode tornar impraticável a solução de diversos problemas, por isto há um interesse grande no uso de heurísticas para resolução de problemas de grande porte.

Como frequentemente a base de heurística é a experiência e a intuição, elas podem falhar. As heurísticas usam informação limitada, não podendo prever como será o espaço de soluções mais adiante na busca. Por isto, uma heurística pode levar um algoritmo de busca a uma solução subótima, ou levá-lo a não conseguir encontrar nenhuma solução. Isto é uma limitação da busca heurística, mas que pode ser eliminada por algoritmos de busca eficientes [34, 64].

As heurísticas podem ser simples ou complexas. Um exemplo de heurística simples seria um método guloso ou um método de busca local, que pára em um ótimo local, utiliza regras claras para limites ou paradas, e passos padrões a serem seguidos. As heurísticas complexas podem não ter essas regras e geralmente a busca pela melhora de uma solução é calculada até um ponto arbitrário ser alcançado. A seguir será descrito na Seção 2.2.1 a definição e o funcionamento da heurística Busca Local, baseado em [72, 103].

2.2.1 Busca Local

A Busca Local é baseada no método de tentativa e erro. A ideia é simples, e apesar disso, é surpreendente como tem resolvido uma gama de problemas de otimização combinatória [85]. Partindo de uma solução inicial s , é feita uma busca no subconjunto

$N(s) \subset S$, que contém soluções vizinhas de s , e a melhor solução encontrada é escolhida. Esse processo se repete enquanto soluções melhores forem obtidas. Caso nenhuma solução melhor que a atual seja encontrada, tem-se um ótimo local e a busca termina. Embora seja possível que o ótimo local encontrado seja também um ótimo global, isso não é garantido pelo método.

De maneira formal, o algoritmo geral de busca local é dado por:

Definição 2.16 Dada uma instância (S, c) de um problema de otimização, em que S é um conjunto factível e c é o custo do mapeamento, a vizinhança é dada por $N : S \rightarrow 2^S$ em que se busca um ponto $s' \in S$ a partir da sub-rotina:

$$melhorar(s) = \begin{cases} s' \in N(s) \mid c(s') < c(s) \\ s, \text{ se não existe } s' \in N(s) \mid c(s') < c(s) \end{cases}$$

Um conceito importante da busca local é a função de vizinhança, pois ela especifica, para cada solução, quais soluções são em algum aspecto ou propriedade próximas e com isso direciona a busca. Essa proximidade pode ser medida pela quantidade de variáveis com valores iguais nas soluções, mas também pode ser medida utilizando outras métricas. Em cada iteração ocorre a movimentação pelo espaço de soluções em que um conjunto $N(s)$ é avaliado, e alguma solução $s' \in N(s)$ é escolhida, ou seja, todo vizinho $s' \in N(s)$ é alcançado pela solução s através da operação chamada de **movimento**. Na Figura 2.2, para uma solução s , existem alguns vizinhos de s representados por s_1, s_2, s_3 e s_4 .

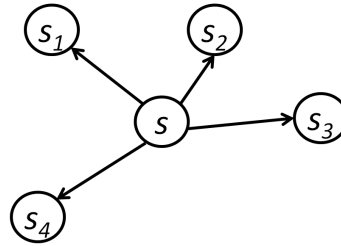


Figura 2.2: Exemplos de possíveis movimentos sobre a vizinhança de s .

O procedimento geral da busca local é mostrado no Algoritmo 2.1. É um algoritmo básico de busca local, conhecido como algoritmo de melhoria iterativa (*Iterative Improvement Algorithm* ou *Hill Climbing Algorithm*). Primeiro, o algoritmo inicia com uma solução inicial factível $s' \in S$ que pode ser gerada aleatoriamente ou criada através de uma heurística construtiva. A partir de uma solução inicial, a cada iteração o algoritmo busca na vizinhança da solução atual uma solução com menor custo. Se uma solução é encontrada, ela se torna a atual solução e o processo continua, caso contrário, a solução atual é um ótimo local e o algoritmo pára.

Algoritmo 2.1: Melhoria Iterativa

```

1  $s \leftarrow$  alguma solução inicial;
2 repita
3    $\text{gerar } s' \in N(s);$ 
4   se  $c(s') < c(s)$  então
5      $s \leftarrow s'$ 
6 até  $c(s') \geq c(s)$  para todo  $s' \in N(s);$ 

```

Para aplicar este método a um problema particular, deve-se fazer uma certa quantidade de escolhas. Primeiro, deve-se obter uma solução inicial factível. Pode-se encontrar várias soluções de diversas maneiras e então escolher o melhor resultado. Em alguns casos, deve-se também decidir quantas soluções iniciais serão calculadas.

Segundo, deve-se escolher uma “boa” vizinhança para o problema e um método de busca. Esta escolha é geralmente guiada por intuição, porque existe teoria para guiar a busca da solução. Uma vizinhança grande parece garantir um ótimo local bom, mas pode demorar muito tempo para encontrá-lo. Também pode ocorrer que, ao se avaliar a vizinhança de uma solução, várias soluções promissoras são encontradas. Para resolver isto pode-se usar alguma **regra de seleção**, para decidir qual solução deve ser avaliada em primeiro lugar.

Existem duas regras de seleção bem conhecidas: **Primeira Melhoria** e **Melhor Melhoria**. A primeira abordagem seleciona a primeira solução de menor custo em relação à solução atual encontrada, termina a iteração e o restante da vizinhança não é avaliada, na mesma iteração. Os vizinhos podem ser gerados aleatoriamente ou em alguma ordem específica. Na segunda abordagem, o funcionamento é contrário à primeira, pois são avaliadas todas as soluções vizinhas, e após é escolhida a solução que traga a melhor melhoria, e assim prossegue a busca.

Estas regras, que permitem a movimentação pela vizinhança podem levar o algoritmo a encontrar boas soluções. Todavia, pode ocorrer que esta movimentação não seja suficiente para garantir e encontrar boas soluções, levando a busca para ótimos locais de baixa qualidade. Um exemplo de vizinhança é a $k\text{-opt}$, definida por $N_k(s) = \{s' : s' \in S \text{ e } s' \text{ pode ser obtida de } s \text{ da seguinte maneira: remova } k \text{ arestas do caminho e as substitua por outras } k \text{ arestas}\}$ [61]. A Figura 2.3 mostra um exemplo de um movimento 2-opt para o Problema do Caixeiro Viajante. Em 2.3(a), um possível circuito para uma instância. Em 2.3(b), um outro circuito, derivado de (a), resultante da aplicação do movimento 2-opt com a troca das arestas (1,2) e (3,4) por (1,3) e (2,4).

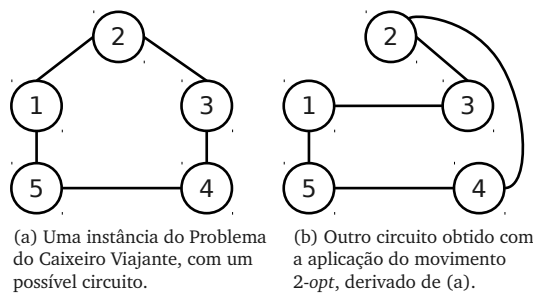


Figura 2.3: Exemplo de movimento k -opt para o Problema do Caixeiro Viajante, com $k = 2$.

2.3 Métodos Metaheurísticos

Osman e Laporte [83] definem **metaheurística** como um processo que guia uma heurística subordinada, com a combinação de diferentes conceitos para explorar e aproveitar o espaço de busca, utilizando estratégias de aprendizagem para organizar a informação a fim de encontrar de maneira eficiente soluções próximas de ótimas ou ótimas. Em geral são utilizadas para tentar resolver problemas complexos de otimização, para os quais métodos exatos não conseguirem ser eficientes.

Metaheurísticas podem ser entendidas como métodos gerais para resolução de problemas, ou seja, são independentes de problemas específicos. Portanto, uma metaheurística pode ser utilizada na resolução de diversas classes de problemas, apenas deve ser refinada para atender as necessidades do problema em questão.

De acordo com Hansen e Mladenović [45, 46], uma metaheurística é um processo de refinamento da busca de uma solução para um problema, que organiza e direciona heurísticas, combinando diferentes conceitos, tentando evitar a parada em um ótimo local. As metaheurísticas, em geral, construídas são complexas e fazem uso de diversos parâmetros, os quais podem levá-las a uma melhor eficiência, porém muitas vezes esta complexidade pode tornar difícil o próprio entendimento dos parâmetros que a tornam eficiente. Estes autores citam algumas propriedades desejáveis nas metaheurísticas, que podem ser aplicadas à heurísticas no geral:

Simplicidade: A metaheurística deve ser baseada em um princípio simples e claro, que pode ser aplicado amplamente;

Precisão: Os passos de uma metaheurística devem ser formulados matematicamente, independente da analogia com princípios da biologia ou física responsável pela inspiração inicial;

Coerência: Todos os passos da heurística devem ser seguidos quando resolver um problema em particular;

Eficiência: Heurísticas para problemas particulares devem fornecer uma solução ótima ou sub-ótima para todas ou pelo menos as instâncias que contém dados reais;

Efetividade: Heurísticas para problemas particulares devem gastar um tempo computacional viável para fornecer soluções ótimas ou sub-ótimas;

Robustez: O desempenho da heurística deve ser consistente sobre uma grande quantidade e variedade de instâncias, e não apenas aperfeiçoá-la para um único conjunto de instâncias;

Amigável: Heurísticas devem ser claramente descritas, fáceis de entender e fáceis de usar. Isto implica em ter poucos parâmetros, e se possível nenhum;

Inovação: Preferivelmente, os princípios das metaheurísticas, eficiência e efetividade, devem conduzir a novos tipos de aplicações.

As metaheurísticas podem ser classificadas em: metaheurísticas de construção e de melhoria, também chamadas de refinamento [103, 111].

As **metaheurísticas de construção** geram uma solução adicionando componentes individuais, como por exemplo, nós, arcos, variáveis, um por vez até que uma solução factível seja obtida. Cada componente é inserido de acordo com a função de avaliação adotada, que depende do problema a ser resolvido. Geralmente, os componentes são escolhidos por uma função gulosa, que procura o melhor componente a ser inserido em cada passo.

As **metaheurísticas de melhoria**, também são chamadas de técnicas de busca local, iniciam com uma solução factível e então a melhoram com uma sequência de passos, como intercalações ou trocas na vizinhança. A solução inicial pode ser obtida por uma heurística construtiva ou gerada aleatoriamente. Um exemplo seria, a busca na vizinhança *k-opt* para o Problema do Caixeiro Viajante.

A principal diferença entre os métodos metaheurísticos é o modo como o método faz para sair de ótimos locais. Podem ser definidos em duas categorias principais, que dizem respeito a como é feita a busca no espaço de soluções: **busca local** e **busca populacional**. Nas metaheurísticas de busca local o espaço de soluções é explorado através de movimentos, que são aplicados na solução corrente a cada iteração, sempre tentando encontrar um vizinho s' melhor que s . Alguns exemplos desta categoria são *Variable Neighborhood Search*, *Simulated Annealing*, Busca Tabu, GRASP. Já o métodos baseados em busca populacional, mantém um conjunto de soluções consideradas “boas”, para combiná-las com o objetivo de produzir soluções melhores. Alguns exemplos, que serão explicados neste trabalho são Algoritmos Genéticos e Colônia de Formigas.

Como as metaheurísticas foram amplamente estudadas para diversos problemas clássicos, elas acabaram por se tornar restritivas. Na busca de bons resultados e melhor desempenho na resolução de problemas complexos, são desenvolvidas as chamadas **metaheurísticas híbridas**, que equivalem à combinação de uma metaheurística com

outras. Em vez de aplicar os **métodos puros** (somente Busca Tabu, por exemplo), são combinadas metaheurísticas com heurísticas, com outras metaheurísticas, e estas podem ser projetadas de maneira sequencial ou entrelaçada. Na sequencial, uma metaheurística é aplicada a um problema, depois outra é aplicada. Na entrelaçada, uma metaheurística é inserida em outra [93].

Nas Seções 2.3.1, 2.3.2, 2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7 e 2.3.8 serão dadas as definições e o funcionamento das metaheurísticas *Variable Neighborhood Search*, *Simulated Annealing*, Busca Tabu, GRASP, Algoritmos Genéticos, Colônia de Formigas, *Path Relinking* e Times Assíncronos, respectivamente, baseadas em [72, 103], dentre outros citados nas seções.

2.3.1 *Variable Neighborhood Search*

A *Variable Neighborhood Search* - VNS (ou Busca em Vizinhança Variável) é uma metaheurística proposta por Hansen e Mladenović em 1997 que baseia-se no princípio de mudança sistemática da vizinhança dentro da busca, tanto na descida para mínimos locais quanto na fuga destes. As definições a seguir sobre VNS são baseadas em [74, 45, 46]. VNS explora sistematicamente os seguintes pontos:

- Um ótimo local com relação a uma dada estrutura de vizinhança não é considerado necessariamente um ótimo local em relação a uma outra estrutura de vizinhança;
- Um ótimo global corresponde a um ótimo local para todas as estruturas de vizinhança;
- Para muitos problemas, ótimos locais, com relação a uma ou mais estruturas de vizinhança, são relativamente próximos.

O último item, de natureza empírica, indica que um ótimo local frequentemente fornece algum tipo de informação sobre o ótimo global. Este é o caso em que os ótimos local e global compartilham muitas variáveis com o mesmo valor, o que sugere uma investigação sistemática da vizinhança de um ótimo local até a obtenção de uma nova solução de melhor valor.

O algoritmo VNS trabalha com várias vizinhanças. Portanto, seja N_k o conjunto finito de estruturas de vizinhanças pré-selecionadas, com $(k = 1, \dots, k_{max})$ e $N_k(s)$ o conjunto de soluções na k -ésima vizinhança de s . Além do conjunto N_k de vizinhanças, usa-se a função de avaliação f , a ser minimizada. Um solução ótima s (um mínimo global) é uma solução viável de tal maneira que para cada solução viável $s' \in S$, tem-se que $f(s) < f(s')$.

Geralmente os algoritmos de busca local utilizam somente uma vizinhança, como no Algoritmo 2.1, ou seja, $k_{max} = 1$. Quando usa-se mais de uma vizinhança, surgem

algumas questões: qual N_k poderia ser usada e quantas usar? Como funciona a ordem de busca? Qual estratégia de busca poderia ser usada nos movimentos da vizinhança? O Algoritmo 2.2 aborda estas questões.

O Algoritmo 2.2 inicia com a seleção de um conjunto de vizinhanças e a geração de uma solução inicial. A partir daí, a cada iteração, o algoritmo busca na vizinhança N_k , da solução atual, uma solução com menor custo. Se a solução é encontrada, ele atualiza a solução e continua a busca na vizinhança N_k , caso contrário ele vai pra próxima vizinhança N_{k+1} . O algoritmo pára de acordo com o critério de parada estabelecido, o qual pode ser, por exemplo: o máximo tempo de computação permitido, número máximo de iterações, número máximo de iterações sem melhoria ou tempo máximo de CPU permitido.

Algoritmo 2.2: VNS

```

1  Inicialização: Selecione conjunto de
   vizinhanças  $N_k$  ( $k = 1, \dots, k_{max}$ ) que será usado
   na “descida”;
2  Encontre uma solução inicial;
3   $k \leftarrow 1$ ;
4  repita
5      Encontre o melhor vizinho  $s' \in N_k(s)$ ;
6      se  $f(s') < f(s)$  então
7           $s \leftarrow s'$ ;
8      senão
9           $k \leftarrow k + 1$ ;
10 até  $k = k_{max}$ ;
```

De fato, o VNS básico está em “descida”, pois a melhor solução atual só pode ser substituída caso seja encontrada outra solução de menor custo, o chamado problema de minimização. Mas é possível transformá-lo num método de “descida-subida” mudando a condição da linha 6 do Algoritmo 2.2: fazer $s \leftarrow s'$ com uma probabilidade p , mesmo que esta solução seja pior que s ; ou pode-se fazer um movimento para a melhor vizinhança k^* entre todas as k_{max} . Pode-se também encontrar o s' no passo 5 como a melhor dentre b (um parâmetro) soluções geradas na k -ésima vizinhança, ou ainda introduzir os parâmetros k_{min} e k_{step} , parâmetros que controlam o processo de mudanças das vizinhanças: no algoritmo anterior trocar $k \leftarrow 1$ por $k \leftarrow k_{min}$ (linha 4) e trocar $k \leftarrow k + 1$ por $k \leftarrow k + k_{max}$ (linha 10).

O algoritmo têm algumas variações tais como: *Variable Neighborhood Descent* (VND), que é mais determinístico, *Reduced Variable Neighborhood Search* (RVNS), que é estocástico. Também existem algumas extensões do VNS: *Skewed Variable Neigh-*

borhood Search (SVNS), que resolve o problema saindo de grandes vales. Para grandes instâncias, o algoritmo *Variable Neighborhood Decomposition Search* (VNDS) é implementado em dois níveis, intercalando o VNS com aproximações sucessivas [45, 46].

2.3.2 *Simulated Annealing*

Simulated Annealing foi criado por Kirkpatrick et al. [58] em 1983. É uma técnica baseada em busca local, em que a busca é feita de maneira probabilística, pois utiliza probabilidades para escolher uma solução vizinha. O objetivo de se utilizar probabilidades está em tentar escapar de ótimos locais de baixa qualidade. Essa técnica também fundamenta-se em uma analogia com a termodinâmica, em um processo chamado recozimento físico de sólidos, que simula o resfriamento de um conjunto de átomos aquecidos. Foram utilizadas [72, 27, 58] como referências básicas dessa subseção.

Como *Simulated Annealing* trabalha de forma probabilística, considere uma heurística de melhoria, onde uma solução vizinha de s é selecionada uniformemente de maneira aleatória na vizinhança $N(s)$. Então, toda solução vizinha $s' \in N(s)$ tem a probabilidade $\frac{1}{|N(s)|}$ de ser escolhida, sendo que a primeira solução s' gerada de melhor custo que s é aceita, ou seja, a solução s é substituída por s' . Se nesta busca aleatória soluções não-melhores forem aceitas, o método provavelmente conseguirá extrapolar a busca para além do primeiro ótimo local encontrado, contudo, a convergência em direção a bons ótimos locais ficará comprometida. Para resolver este problema de convergência utiliza-se um fator de deterioração da qualidade da solução, que não deve ultrapassar um determinado limite. Este limite é chamado de Δ e é dado por $\Delta = f(s') - f(s)$.

O Algoritmo 2.3 descreve o funcionamento do *Simulated Annealing*. Primeiro, é gerada uma solução inicial (linha 1). A cada iteração é gerado aleatoriamente um único vizinho s' da solução corrente s (linha 6). Suponha um problema de minimização, Δ equivale a variação do valor da função objetivo quando é realizado um movimento na vizinhança, ou seja, $\Delta = f(s') - f(s)$. Se $\Delta < 0$ (linha 8), o movimento é aceito e a solução vizinha passa a ser a nova solução atual. Caso $\Delta \geq 0$ (linha 11), s' também pode ser aceita, mas com probabilidade $e^{-\frac{\Delta}{T}}$, e T é um parâmetro do método que equivale a temperatura e que regula a probabilidade de aceitar soluções de pior custo.

A temperatura T inicia com um valor alto T_0 (linha 3). Após uma quantidade fixa de iterações, que representam quantas iterações são necessárias para o sistema atingir o equilíbrio térmico em uma dada temperatura, a temperatura é diminuída gradativamente por uma razão de resfriamento α , tal que $T_k \leftarrow \alpha \times T_{k-1}$, e $0 < \alpha < 1$ (linha 15). Este procedimento faz com que diminua a chance de entrar em mínimos locais. À medida em que T se aproxima de zero, o algoritmo comporta-se como o método de descida porque diminui a probabilidade de se aceitar movimentos de piora, pois como $T \rightarrow 0$, tem-se

também que $e^{\frac{-\Delta}{T}} \rightarrow 0$).

Algoritmo 2.3: *Simulated Annealing*

```

1   $s \leftarrow$  Uma solução inicial;
2   $i \leftarrow 1$  /* Quantidade de iterações na temperatura T */
3   $T \leftarrow T_0$  /* Temperatura atual */
4  enquanto  $T > 0$  faça
5      repita
6          Gere um vizinho qualquer  $s' \in N(s)$   $\Delta \leftarrow f(s') - f(s)$ ;
7          se  $\Delta < 0$  então
8               $s \leftarrow s'$ ;
9          senão
10             se  $e^{\frac{-\Delta}{T}} > \text{random}[0,1]$  então
11                  $s \leftarrow s'$ ;
12              $i \leftarrow i + 1$ ;
13     até  $i < \text{máximo de iterações}$  /* Critério de parada */
14     ;
15      $T \leftarrow \alpha \times T$ ;
16      $i \leftarrow 0$ ;

```

2.3.3 Busca Tabu

O método Busca Tabu foi criado a partir de trabalhos independentes de Glover [37] e Hansen [44]. A diferença entre Busca Local e Busca Tabu é a maneira de explorar a vizinhança, pois na primeira as soluções melhores que a atual são escolhidas da vizinhança e na segunda a melhor solução vizinha é escolhida. A estratégia de escolher o melhor vizinho (*best improvement*), diferentemente de *Simulated Annealing*, junto com uma estrutura de memória para armazenar as soluções geradas, tem como objetivo não deixar a busca presa em um ótimo local.

O algoritmo trabalha da seguinte maneira: inicia com uma solução inicial s_0 , e a cada iteração do algoritmo um subconjunto V da vizinhança de $N(s)$ é explorado ($V \subset N(s)$). A solução vizinha $s' \in V$ com melhor valor da função objetivo f é selecionado para solução atual s , mesmo que s' seja pior que s , isto é, que $f(s') > f(s)$.

Este critério de escolha do melhor vizinho é utilizado para escapar de ótimos locais. No entanto, isto pode fazer com que o algoritmo retorne à soluções já exploradas, fazendo com que ele cicle. Para que isso não ocorra, usa-se uma *lista tabu* T , uma lista que possui soluções visitadas recentemente e que ficam proibidas de serem visitadas por

uma certa quantidade de iterações, evitando dessa maneira, caminhos cíclicos no grafo de vizinhança.

Na estrutura clássica, a lista possui movimentos reversos aos últimos $|T|$ movimentos realizados ($|T|$ é um parâmetro do método) e trabalha como uma fila de tamanho fixo, ou de seja, quando um novo movimento é adicionado à lista e ela está cheia, o movimento mais antigo sai. Dessa forma, quando o subconjunto $V \subset N(s)$ é explorado os m elementos que estão na lista tabu T são excluídos da busca, isto é, os vizinhos s' que são obtidos dos m movimentos em T são excluídos da busca.

O Algoritmo 2.4 apresenta o procedimento básico da Busca Tabu. Iniciando com uma solução inicial e a lista tabu T (linhas 1 e 2), a melhor solução s' não-tabu é escolhida (linha 5) na vizinhança de s é escolhida para a busca na próxima iteração. Esta solução é inserida na lista tabu (linha 9), de modo que ela não seja utilizada novamente durante uma certa quantidade de iterações. Ainda na linha 9, a atualização da lista tabu trata da adição de novas soluções e a manutenção do tamanho da lista. O tamanho do lista pode influenciar na busca, no sentido de que se for uma lista muito pequena pode não evitar ciclos e se for uma lista muito grande pode limitar o espaço de soluções, restringindo a busca de forma exagerada.

Algoritmo 2.4: Busca Tabu

```

1  $T \leftarrow \emptyset$  /* Lista Tabu                                     */
2  $s \leftarrow$  alguma solução inicial;
3 repita
4    $\quad$  encontrar a melhor  $s' \in N(s) \setminus T$  ;
5   se  $f(s') < f(s)$  então
6      $\quad$   $s \leftarrow s'$ ;
7    $s \leftarrow s'$ ;
8    $\quad$  Atualize a lista tabu  $T$ ;
9 até critério de parada;
```

A lista tabu pode eliminar ciclos, pois seu objetivo é garantir o não retorno de uma solução já visitada anteriormente por $|T|$ iterações. Mas isto pode fazer com que movimentos para soluções que ainda não foram visitadas não sejam alcançados (alguns movimentos são proibidos, de acordo com a lista tabu). Por isto, uma função de aspiração é utilizada. O Algoritmo 2.5 mostra a aplicação desta função, que retira, sob certas circunstâncias, o status tabu de um movimento. Para cada possível valor v da função objetivo existe um nível de aspiração $A(v)$, em que uma solução $s' \in V$ pode ser gerada se $f(s') < A(f(s))$, mesmo o movimento m esteja na lista tabu. Para cada valor v da função objetivo, a função de aspiração retorna um valor $A(v)$ que representa o valor

que o algoritmo aspira para chegar em v . Uma aplicação desta ideia seria por exemplo, considerar $A(f(s) = f(s^*))$, onde s^* é a solução encontrada até o momento. Neste caso, o movimento tabu m é aceito se ele conduzir a um vizinho melhor do que s^* . Esta é a aspiração do objetivo, que se baseia no fato de que soluções melhores que a solução s^* atual, mesmo que geradas por movimentos tabu, ainda não foram visitadas, devido ao fato que a lista de movimentos tabu pode tanto impedir o retorno de uma solução gerada anteriormente, quanto outras soluções ainda não geradas.

Em relação ao critério de parada, duas regras são utilizadas para interromper o procedimento (linha 6). Na primeira o algoritmo pára quando é atingida uma determinada quantidade de iterações sem melhora no valor da solução. Na segunda, quando o valor da melhor solução alcança (ou chega próximo de) um limite inferior conhecido. Este critério evita a execução desnecessária do algoritmo, desde que se tenha alcançado uma solução próxima da ótima ou avaliada como suficientemente boa.

Os principais parâmetros do Algoritmo 2.5 são a cardinalidade $|T|$ da lista tabu, a função de aspiração A , a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração, e $BTmax$, a quantidade máxima de iterações sem melhora no valor da melhor solução. O algoritmo pode utilizar o parâmetro f_{min} , que equivale a um limite inferior, isto é, o valor mínimo conhecido de f , que em alguns casos pode ser conhecido.

Algoritmo 2.5: Busca Tabu com Função de Aspiração

```

1  $s^* \leftarrow$  Melhor solução obtida até o momento ;
2  $i \leftarrow 0$  /* Quantidade de iterações */
3  $melhorIter \leftarrow 0$  /* Iteração mais recente que forneceu  $s^*$  */
4  $T \leftarrow \emptyset$  /* Lista Tabu */
5 Inicialize a função de aspiração  $A$  ;
6 enquanto  $(i - melhorIter \geq BTmax)$  e  $(f(s) > f_{min})$  faça
7    $i \leftarrow i + 1$ ;
8   Seja  $s' \leftarrow s \oplus m$  o melhor elemento de  $V \subseteq N(s)$  tal que o movimento  $m$  não
   seja tabu ( $m \notin T$ ) ou  $s'$  atenda a condição de aspiração  $f(s') < A(f(s))$ ;
9   Atualize a lista tabu  $T$ ;
10   $s \leftarrow s'$ ;
11  se  $f(s) < f(s^*)$  então
12     $s^* \leftarrow s$ ;
13     $melhorIter \leftarrow i$ ;
14  Atualize a função de aspiração  $A$ ;
15  $s \leftarrow s^*$ ;
16 Retorne  $s$ ;
```

No processo de busca existem alguns procedimentos que podem melhorar a

busca por melhores soluções. Existem as chamadas estratégias de intensificação, que têm por objetivo explorar a pesquisa em determinadas regiões que são consideradas promissoras. Um exemplo de estratégia de intensificação é retornar à uma solução já visitada para explorá-la de maneira mais efetiva. Um critério de término deve ser estipulado para encerrar o período da intensificação.

Os métodos baseados em Busca Tabu ainda trabalham com estratégias de diversificação. Esta estratégia é utilizada para redirecionar a pesquisa para regiões ainda não suficientemente exploradas do espaço de soluções. Geralmente utilizam uma memória de longo prazo. Ao contrário das estratégias de intensificação, as estratégias de diversificação procuram gerar soluções que têm atributos significativamente diferentes daqueles encontrados nas melhores soluções obtidas. A diversificação é utilizada de modo geral, somente em determinadas situações. Um exemplo seria, dada uma solução s e não existindo movimentos m de melhora para ela, isto é, o espaço de busca daquela região foi esgotado, é estabelecida uma penalidade $w(s, m)$ para o uso destes movimentos. Também é utilizado para acionar a diversificação, o parâmetro que armazena a quantidade fixa de iterações sem melhora no valor da solução ótima corrente.

Muitos trabalhos explicam esta técnica. Dentre eles, podemos citar: [72, 22, 35, 38, 39, 49, 50].

2.3.4 GRASP

A técnica GRASP (*Greedy Randomized Adaptive Search Procedure* - Procedimento de busca adaptativa gulosa e randômica) foi proposta por Feo e Resende [31]. É uma técnica baseada no processo chamado *Multistart*, uma estratégia em que o algoritmo é processado múltiplas vezes, com o objetivo de obter ótimos locais de melhor qualidade. Uma trivial extensão do algoritmo de Melhoria Iterativa baseado nesta ideia é o algoritmo de Múltiplas Inicializações Aleatórias (*Random Restart*). O algoritmo básico é executado várias vezes, a partir de uma solução inicial aleatória e diferente das já utilizadas. Como o algoritmo de Melhoria Iterativa é fortemente independente da solução inicial, ele pode ter dificuldade de alcançar ótimos locais, e GRASP tenta solucionar este problema gerando a cada iteração soluções melhores.

O Algoritmo 2.6 mostra o funcionamento de GRASP, de maneira genérica. O método consiste de duas fases: uma fase de construção em que uma solução gulosa e aleatória é gerada (linha 3), e de uma fase de busca local (linha 4), em que é pesquisado um ótimo local a partir da solução encontrada na fase de construção. Na linha 5 a melhor solução é atualizada. Perceba que diferentes ótimos locais são encontrados, e a melhor solução encontrada após todas as iterações realizadas é retornada como resultado (linha 9). As soluções iniciais são geradas de forma aleatória e gulosa com o objetivo de

aumentar a variedade de soluções.

Algoritmo 2.6: GRASP

```

1  $f^* \leftarrow \infty$ ;
2 repita
3    $s \leftarrow \text{Construção}()$ ;
4    $\text{BuscaLocal}(s)$ ;
5   se  $(f(s) < f(s^*))$  então
6      $s^* \leftarrow s$ ;
7 até critério de parada;
8  $s \leftarrow s^*$ ;

```

A seguir, são detalhadas cada fase do GRASP. Na fase de construção, uma solução é construída elemento por elemento, em cada iteração. A cada iteração da fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista C de candidatos, seguindo um critério de ordenação pré-determinado. Esta seleção é baseada em uma função adaptativa gulosa $g : C \rightarrow \mathbb{R}$ que estima o benefício da seleção de cada um dos elementos. Como os benefícios de cada elemento são atualizados a cada iteração, logo as próximas soluções conterão algumas características da solução anterior. Isto é chamado de **heurística adaptativa**. Como cada elemento que compõe a lista de candidatos é selecionado de maneira aleatória em um subconjunto formado pelos melhores elementos que compõem a lista de candidatos, ele é considerado probabilístico. Este subconjunto é chamado de **lista de candidatos restrita** (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP.

O Algoritmo 2.7 ilustra o procedimento de construção da solução inicial. O parâmetro $\alpha \in [0, 1]$ controla o nível de gulosidade e aleatoriedade do procedimento *Construção*. Se $\alpha = 0$ as soluções geradas são totalmente gulosas, e se $\alpha = 1$ as soluções geradas são totalmente aleatórias. Ele é o único parâmetro que deve ser ajustado em GRASP. Quando α possui um valor que se aproxima da escolha aleatória, há uma grande diversidade de soluções construídas, mas muitas destas soluções podem ser de qualidade baixa, e consequentemente tornar o processo da busca local mais lento. Já quando α possui um valor que se aproxima da escolha gulosa, as soluções finais acabam tendo uma qualidade muito próxima a que foi obtida de forma gulosa, entretanto, não ocorre tanta diversidade nas soluções construídas.

Algoritmo 2.7: Construção

```

1  $s \leftarrow \emptyset$ ;
2 Inicialize o conjunto de candidatos C;
3 enquanto  $C \neq \emptyset$  faça
4    $g(t_{min}) = \min\{g(t) \mid t \in C\}$ ;
5    $g(t_{max}) = \max\{g(t) \mid t \in C\}$ ;
6    $LCR = \{t \in C \mid g(t) \leq$ 
       $g(t_{min}) + \alpha(g(t_{max}) - g(t_{min}))\}$ ;
7   Selecione, aleatoriamente, um elemento
       $t \in LCR$ ;
8    $s \leftarrow s \cup \{t\}$ ;
9   Atualize o conjunto C de candidatos;

```

O Algoritmo 2.8 de busca local utilizado em GRASP, obterá boas soluções caso a solução inicial também seja boa. Então, quando o algoritmo de busca local inicia com uma solução boa, ele tende acelerar seu processo de busca. Portanto, o algoritmo de construção é importante, pois ele irá implica profundamente no resultado final.

Algoritmo 2.8: Busca Local GRASP

```

1  $V = \{s' \in N(s) \mid f(s') < f(s)\}$ ;
2 enquanto  $|V| > 0$  faça
3   Selecione  $s' \in V$ ;
4    $s \leftarrow s'$ ;
5    $V = \{s' \in N(s) \mid f(s') < f(s)\}$ ;
6 Retorne s;

```

Como o algoritmo GRASP trabalha com aspectos aleatórios e gulosos, é importante saber balanceá-los. O trabalho de Prais e Ribeiro [92] trata da variação de parâmetros em procedimentos GRASP.

2.3.5 Algoritmos Genéticos

Os Algoritmos Genéticos foram desenvolvidos por Holland e seus colegas nos anos 70. São algoritmos de busca baseados em processos de seleção natural de evolução, da Teoria de Darwin, em que os indivíduos com características melhores têm maiores chances de sobrevivência e de produzirem filhos cada vez mais aptos, e os indivíduos mais fracos (menos aptos) tendem a desaparecer. Para esta seção, são referenciados [41, 71].

Alguns elementos de Algoritmos Genéticos são mostrados na Figura 2.4. A **população** é um conjunto de **indivíduos**, também chamados de **cromossomos**. Cada cromossomo equivale a uma solução do problema. Dessa forma, uma população é um conjunto de soluções. O cromossomo é dividido em componentes, chamados **alelos**, que são os possíveis valores que cada componente da solução pode assumir. O valor que cada alelo possui é chamado de **gene**. Um mecanismo de reprodução baseado em processos evolutivos é aplicado sobre a população, com o objetivo de explorar o espaço de busca e encontrar as melhores soluções para o problema. Cada indivíduo é avaliado por uma **função de aptidão** (a função objetivo), a qual mensura seu grau de adaptação ao meio. Quanto maior for o valor da função de aptidão, quer dizer que mais o indivíduo está adaptado ao meio.

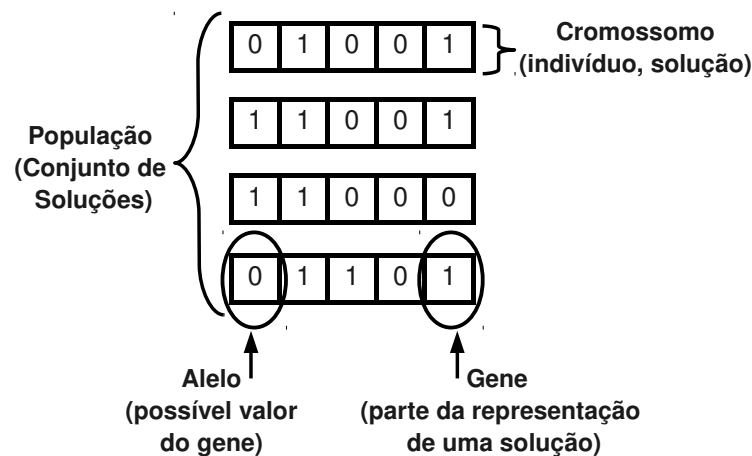


Figura 2.4: Elementos de um Algoritmo Genético.

Dessa forma, para um problema particular, um Algoritmo Genético (como qualquer algoritmo evolutivo) deve ter estes cinco componentes:

- Uma representação genética de potenciais soluções par o problema;
- Um modo de criar uma população de soluções inicial;
- Uma função de avaliação que segue as regras do ambiente, avaliando as soluções em relação à sua "aptidão";
- Operadores genéticos que alteram a composição dos filhos;
- Valores de vários parâmetros que um Algoritmo Genético usa (tamanho da população, probabilidade da aplicação dos operadores genéticos).

O algoritmo básico funciona da seguinte maneira: inicia sua busca com uma população $\{s_1^0, s_2^0, \dots, s_n^0\}$, em que s_i^j , com $i = 1, \dots, n$, equivale à uma solução, e $j = 1, \dots, t$,

que equivale ao tempo em que a solução foi criada. A população inicial, como $j = 0$, é chamada de população no tempo 0.

A Figura 2.5 ilustra o procedimento principal de um Algoritmo Genético, um ciclo que cria uma população $\{s_1^{t+1}, s_2^{t+1}, \dots, s_n^{t+1}\}$ no tempo $t + 1$ a partir de uma população gerada no tempo t . Para atingir este objetivo, são selecionados dois indivíduos da população de tempo t , chamados pais (a), que passam por uma fase de reprodução através da seleção de um ponto de corte (b), e um processo de recombinação (*crossover*)(c), são gerados os filhos (*offsprings*), que podem passar por um processo de mutação (d), onde um determinado gene pode ser modificado.

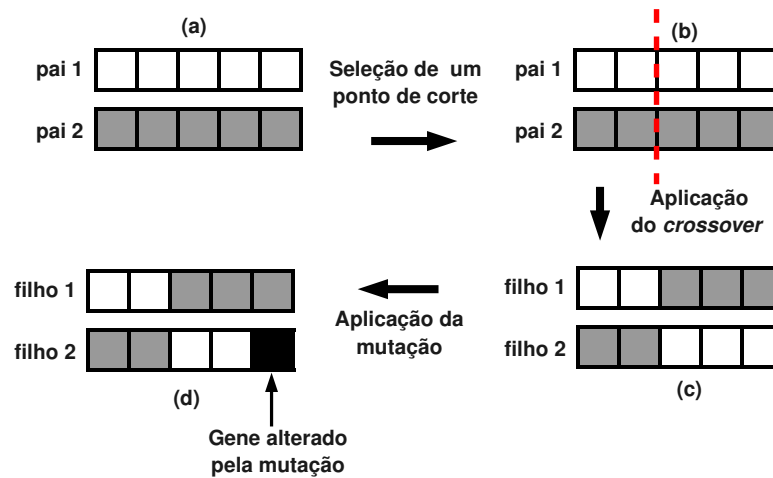


Figura 2.5: Procedimento Básico de um Algoritmo Genético.

Existem várias formas de selecionar indivíduos para o processo de **reprodução**. Uma delas é a *Binary Tournament Selection*. Neste processo, os indivíduos são selecionados aleatoriamente e aquele que tiver o maior valor para a função de aptidão é escolhido para ser o pai, de forma análoga, o segundo pai é escolhido. Pode-se também selecionar os pais aleatoriamente.

Depois de feita a seleção dos pais, é aplicada uma **operação de recombinação** neles e então gerados filhos (geralmente dois). Nesta operação, os genes dos pais são combinados de forma que cada filho, há um conjunto de genes de cada um dos cromossomos pais, como no passo (b) para (c) da Figura 2.5. A operação de mutação consiste em alterar aleatoriamente uma parte dos genes de cada cromossomo (componentes da solução). As operações de recombinação e mutação são realizadas com uma certa probabilidade.

Depois de gerada uma nova população no tempo $t + 1$, define-se a população sobrevivente, ou seja, as n soluções que integrarão a nova população. A população sobrevivente é definida pela aptidão dos indivíduos. Os critérios em geral para escolher os cromossomos sobreviventes são os seguintes: **aleatório**; **roleta** (a chance de sobrevivência de cada cromossomo é proporcional ao seu nível de aptidão); **misto** (combinação dos

dois critérios anteriores). Nesses critérios admite-se a sobrevivência dos indivíduos mais aptos. O objetivo do uso destes critérios é escapar de ótimos locais.

O algoritmo 2.9 apresenta o pseudocódigo de um Algoritmo Genético básico. Os principais parâmetros do algoritmo são o tamanho n da população, a probabilidade e o ponto de corte da operação *crossover*, a probabilidade de mutação, a quantidade de gerações e a quantidade de gerações sem melhora. O tempo inicia em zero (linha 1), depois é gerada uma população inicial no tempo t (linha 2). A população é avaliada (linha 3). Depois desta avaliação, o algoritmo entra em ciclo, e quando os critérios de qualidade forem satisfatórios o algoritmo pára (linha 4), caso contrário, uma nova população é gerada com base na população anterior (linha 6). A geração da população das linhas 2 e 6, equivalem a selecionar os pais, aplicar *crossover* e aplicar a mutação. Após isto, a população é avaliada pela função de aptidão (linha 7), e definida a população sobrevivente (os mais aptos, com os melhores valores da função de aptidão) (linha 8).

Algoritmo 2.9: Construção

```

1  $t \leftarrow 0$ ;
2 Gere a população inicial  $P(t)$ ;
3 Avalie  $P(t)$ ;
4 enquanto os critérios de parada não estiverem satisfeitos faça
5    $t \leftarrow t + 1$ ;
6   Gere  $P(t)$  a partir de  $P(t - 1)$ ;
7   Avalie  $P(t)$ ;
8   Defina a população sobrevivente;
```

Um cromossomo, que equivale a uma solução do problema, geralmente é construído na forma de um vetor ou lista, $p = (x_1, x_2, \dots, x_m)$, em que cada componente x_i representa um gene (uma parte da solução). Eles podem ser representados como, por exemplo: a **representação binária** (representação clássica) e a **representação por inteiros**.

Na representação binária, uma solução para o problema é representada por um vetor de 0's e 1's. Um exemplo de manipulação desta solução seria a quantidade de 1's presente na solução. Um exemplo, seria maximizar a função $f(x) = |11 \times \text{num}(x) - 150|$, em que $\text{num}(x)$ contém a quantidade de 1's do vetor cromossomo.

Em relação aos operadores, o operador *crossover* clássico efetua cruzamentos entre dois ou mais cromossomos pais para formar cromossomos filhos a partir da união de genes de cada pai. São feitos cortes, que podem ser aleatórios, ou aplicados na parte central dos cromossomos pais. Veja na Figura 2.6, por exemplo, dois cromossomos pais $p_1 = (110001)$ e $p_2 = (100011)$. Ao aplicar um corte no meio de p_1 e p_2 , podem ser gerados dois filhos f_1 e f_2 , cada filho herda uma parte de cada cromossomo pai.

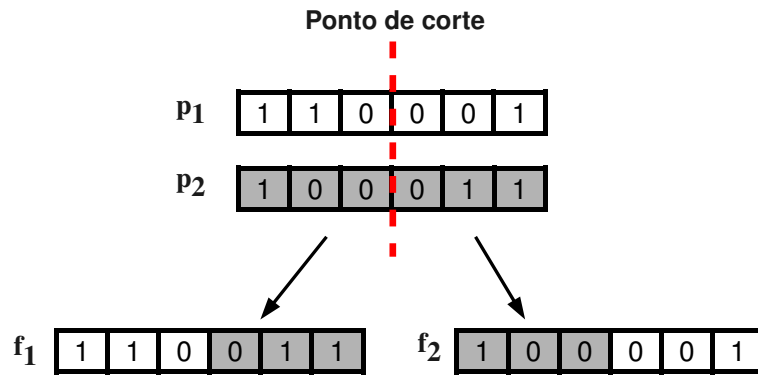


Figura 2.6: *Procedimento Básico de crossover.*

Já o operador de mutação clássico consiste em alterar um ou mais genes de um cromossomo. Por exemplo, seja o cromossomo $p = (110001)$. Uma mutação dele poderia ser alteração do valor 0 para 1, ou vice-versa, resultaria em $p' = (110\ 101)$.

Os operadores de *crossover* e mutação foram citados como clássicos, pois são o caso mais básico do procedimento. Para determinados problemas, aplicar os movimentos básicos pode não gerar uma solução viável. Por isso, para cada problema deve ser analisado qual o procedimento que deve ser aplicado nos operadores de *crossover* e mutação.

2.3.6 Colônia de Formigas

A técnica Colônia de Formigas (*Ant Colony Optimization Metaheuristic - ACO*) foi criada por Dorigo [23] e é inspirada no comportamento das formigas. Como muitas espécies de formigas são quase cegas, a comunicação entre elas é feita através de uma substância química chamada **feromônio**. Em algumas espécies o feromônio é usado para criar caminhos, para guiar as formigas. As formigas saem aleatoriamente da colônia à procura de alimentos. Quando encontram, elas depositam o feromônio no chão, fazendo uma trilha. As formigas sentem o cheiro do feromônio, e escolhem com maior probabilidade o caminho que tem o cheiro mais forte, ou seja, com maior quantidade de feromônio. Estas trilhas são usadas para encontrar a fonte de alimento e achar o caminho de volta. Esta seção foi também baseada em [23, 24, 25].

O método simula o comportamento de um conjunto de agentes, as formigas, que cooperam entre si para resolver um problema de otimização. Perceba que esta é uma técnica construtiva, visto que os caminhos de soluções são construídos pelas formigas através do feromônios. As soluções são construídas de maneira probabilística e utilizam como informação a trilha de feromônio, que muda dinamicamente durante a execução do algoritmo, deixando com feromônio mais forte o caminho mais próximo da solução e a informação heurística específica do problema a ser resolvido.

O componente chave de um algoritmo ACO é um modelo parametrizado probabilístico, que é chamado de modelo de feromônio. O modelo de feromônio consiste em um vetor T de parâmetros (parâmetros de trilha de feromônio). O modelo de feromônio é usado para gerar probabilisticamente soluções para o problema em questão, construindo um conjunto finito de componentes da solução. O algoritmo ACO atualiza em tempo de execução os valores dos feromônios, utilizando soluções geradas anteriormente. A atualização tem como objetivo concentrar a pesquisa em regiões do espaço de busca que contêm soluções de alta qualidade. Em geral, a abordagem do algoritmo ACO tenta resolver um problema de otimização, repetindo os seguintes passos: as soluções candidatas são construídas utilizando um modelo de feromônio, ou seja, uma distribuição de probabilidade parametrizada sobre o espaço de soluções; as soluções candidatas são utilizadas para modificar os valores de feromônio, considerando as características das melhores soluções encontradas.

O comportamento das formigas em um algoritmo ACO pode ser resumido a seguir. Uma colônia de formigas se move de forma concorrente e assíncrona construindo caminhos no espaço de busca, aplicando uma política de decisão local estocástica, que faz uso das trilhas de feromônio e informações heurísticas. Ao se moverem, as formigas constroem novas soluções para o problema de otimização. Construída uma solução, ou durante a construção de uma solução, a formiga avalia a solução (parcial ou completa) e deposita uma trilha de feromônio apenas nas componentes ou conexões usadas durante o caminho. A informação do feromônio é usada para direcionar a busca das outras formigas.

Para melhor compreensão, suponha um problema de minimização. A vizinhança S é definida por um conjunto de variáveis discretas X_i com valores $c_i^j \in D_i = \{c_i^1, \dots, c_i^{|D_i|}\}$, $i = 1, \dots, n$. A variável c_i^j equivale a um componente da solução X_i . O modelo de feromônio consiste no parâmetro rastro de feromônio T_i^j para cada componente c_i^j . O conjunto de todos os componentes da solução é chamado de $\mathcal{C}$. T é o vetor de todos os parâmetros de rastro dos feromônios. Como um problema de otimização combinatória pode ser modelado de diferentes maneiras, então diferentes modelos podem ser usados para definir diferentes modelos de feromônios.

Uma definição geral de ACO é dada pelo Algoritmo 2.10, obtida de [24]. Os parâmetros utilizados pelo algoritmo são: n é a quantidade de formigas, T é o vetor de todos os parâmetros de rastro dos feromônios, s^* é a melhor solução encontrada até o momento, $\mathcal{S}_{iter}$ é o conjunto de soluções que são construídas na iteração corrente. Primeiro, o algoritmo inicializa os valores dos parâmetros de feromônio (linha 1). Os passos a seguir são repetidos até o critério de parada ser alcançado: o conjunto de soluções construídas inicia vazio (linha 4); há um laço que itera a quantidade de formigas (linha 5). Em cada iteração, as n formigas constroem soluções de maneira probabilística para o problema de otimização combinatória em questão, explorando um dado modelo de feromônio (linha

6). A seguir, é aplicado o procedimento de Busca Local às soluções construídas, que é opcional (linha 8). Na linha 9, a solução com melhor valor é atualizada em s^* . Então, antes da próxima iteração iniciar, algumas das soluções são usadas para executar uma atualização nos feromônios (linha 12). Ao final, a melhor solução é retornada (linha 14).

Algoritmo 2.10: Colônia de Formigas

```

1  InicializaValoresFeromônio( $T$ );
2   $s^* \leftarrow NULL$ ;
3  repita
4       $\mathfrak{S}_{iter} \leftarrow \emptyset$ ;
5      para cada  $j = 1, \dots, n$  faça
6          ConstroiSolução( $T$ );
7          se  $s$  é uma solução válida então
8               $s \leftarrow BuscaLocal(s)$  /* Opcional */
9              se  $(f(s) < f(s^*))$  ou  $(s^* = NULL)$  então
10                  $s^* \leftarrow s$ ;
11                  $\mathfrak{S}_{iter} \leftarrow \mathfrak{S}_{iter} + 1$ ;
12  AtualizarFeromônio ( $T, \mathfrak{S}_{iter}, s^*$ );
13  até critério de parada;
14  Retorne  $s^*$ .
```

Os critérios de parada comumente usados são a quantidade máxima de iterações e a estagnação, que é a situação na qual todas as formigas seguem sempre o mesmo percurso, quando há um crescimento excessivo de feromônio em um determinado caminho.

O algoritmo possui quatro funções básicas. A seguir, os detalhes de cada uma:

InicializaValoresFeromônio(T): O algoritmo inicia os valores de feromônios com um valor constante $c > 0$.

ConstroiSolução(T): Uma heurística construtiva reúne soluções como sequências de elementos de um conjunto finito de componentes da solução $\mathcal{C}$. Uma solução parcial s^p inicia vazia, então a cada passo da construção da solução s^p é adicionado um componente viável da solução a partir da $\mathfrak{R}(s^p) \subseteq \mathcal{C} \setminus \{s^p\}$. Este conjunto é determinado, a cada passo de construção, pelo procedimento de construção da solução de tal forma que as restrições do problema sejam satisfeitas.

A escolha de um componente da solução $c_i^j \in \mathfrak{R}(s^p)$, a cada passo da construção, é probabilisticamente feito com relação ao modelo de feromônio. A probabilidade da escolha c_i^j é proporcional a $[\tau_i^j]^\alpha [\eta c_i^j]^\beta$, em que η é uma função que atribui a cada componente de solução válida uma informação heurística. Os valores dos parâmetros α e β , com $\alpha > 0$ e $\beta > 0$, determinam a importância do feromônio e da informação

heurística. A informação heurística é opcional, mas geralmente é necessária para o algoritmo alcançar um bom desempenho.

Busca Local: Uma vez que as soluções foram construídas, antes de atualizar os feromônios, pode-se aplicar ações opcionais, chamadas de *daemon*. As ações *daemon* podem ser utilizadas para implementar ações centralizadas, que não podem ser realizadas pelas formigas de maneira isolada. O *daemon* mais utilizado é a aplicação de Busca Local para construir as soluções, de forma que as melhores soluções são usadas para decidir que feromônios serão atualizados. Apesar de seu uso ser opcional, foi observado experimentalmente que a Busca Local melhora o desempenho global do algoritmo. Outro exemplo de *daemon* seria uma coleção de informações globais que podem ser usadas para decidir se é útil ou não depositar feromônio adicional para guiar a busca sob uma perspectiva não local. O *daemon* pode observar o caminho encontrado por cada formiga da colônia e escolher depositar uma quantidade extra de feromônio apenas nas componentes usadas pela formiga que construiu a melhor solução.

Atualizar Feromônio ($T, \mathfrak{S}_{iter}, s^*$): O objetivo da atualização do feromônio é aumentar os valores dos feromônios que estão associados a boas soluções e diminuir aqueles associados com as soluções ruins. Geralmente, esta atualização é realizada de duas maneiras: diminuindo os valores de feromônio por **evaporação de feromônio**, e pelo aumento das taxas de feromônio associado a um conjunto de boas soluções. A evaporação de feromônio é responsável por diminuir a quantidade de feromônio depositado pelas formigas, e decresce ao longo do tempo.

Muitos algoritmos ACO utilizam a seguinte regra de atualização:

$$\tau_i^j \leftarrow (1 - \rho)\tau_i^j + \frac{\rho}{\mathfrak{S}_{upd}} \sum_{\{s \in \mathfrak{S}_{upd} | c_i^j \in s\}} F(s), \quad (2-1)$$

para $i = 1, \dots, n$ e $j = 1, \dots, |D_i|$. Existem diferentes regras para atualização dos feromônios ($\mathfrak{S}_{upd}$). O parâmetro $\rho \in (0, 1]$ é chamado de **taxa de evaporação**, que tem a função de diminuir uniformemente todos os valores de feromônio. A evaporação de feromônio é necessária para evitar uma convergência muito rápida do algoritmo para uma região subótima. Esta evaporação equivale a uma forma de esquecimento, favorecendo a exploração de novas áreas no espaço de busca.

A função $F : \mathfrak{S} \mapsto \mathbb{R}^+$ é uma função que se $f(s) < f(s')$ então $+\infty > F(s) \geq F(s'), \forall s \neq s' \in \mathfrak{S}$, onde $\mathfrak{S}$ é o conjunto de todas as sequências dos componentes da solução que podem ser construídos pelo algoritmo ACO e que correspondem a soluções factíveis. A função F é chamada de **função de qualidade**.

Mais detalhes sobre ACO podem ser encontrados em [17, 101, 26].

2.3.7 *Path Relinking*

A técnica *Path Relinking* (Reconexão por Caminhos) foi proposta por Glover [40] com o objetivo de explorar trajetórias que conectam soluções de elite, isto é, soluções ótimas ou próximas de ótimas, obtidas por Busca Tabu ou *Scatter Search*¹. Esta seção é baseada em [98].

Para encontrar as melhores soluções são gerados e explorados caminhos no espaço de soluções partindo de uma ou mais soluções de elite que levam a outras soluções de elite. Estas soluções são alcançadas selecionando-se movimentos que introduzem atributos das soluções guia na solução corrente.

Esta técnica pode ser aplicada de acordo com duas estratégias básicas:

- Reconexão por caminhos aplicada como uma estratégia pós-otimização entre todos os pares de soluções de elite;
- Reconexão por caminhos aplicada como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local.

De acordo com Rosseti [98], a aplicação da técnica de reconexão por caminhos como um procedimento de intensificação a cada ótimo local é mais eficaz do que empregá-la como um procedimento de pós-otimização. No caso em que a técnica trabalha bem, ela trabalha da seguinte maneira: o *Path Relinking* é aplicado a pares (s_1, s_2) de soluções, sendo s_1 a solução ótima local corrente obtida após a aplicação de busca local e s_2 é uma solução selecionada aleatoriamente de um conjunto elite que contém as melhores soluções encontradas durante a exploração do espaço de soluções. O conjunto inicia vazio e cada solução obtida no final de uma busca local é uma candidata para o conjunto de elite, desde que ela seja melhor que a solução de pior qualidade desse conjunto e apresente um percentual mínimo de diferença em relação a cada solução de conjunto de elite. Este procedimento evita que o conjunto elite contenha soluções muito parecidas. Se o conjunto estiver vazio, a solução é simplesmente inserida no conjunto. Se o conjunto elite já possui o tamanho máximo definido a solução corrente é candidata a ser inserida neste conjunto e, caso ela seja escolhida, substituirá a solução de pior qualidade.

O Algoritmo 2.11 mostra o funcionamento do algoritmo *Path Relinking*. Na linha 2, g' recebe a melhor solução entre o par de soluções s e g . Na linha 3 é calculada a diferença simétrica $(\Delta(s, g))$ entre s e g . Esta diferença resulta no conjunto de movimentos que deve ser aplicado à solução inicial (s) para alcançar a solução guia g . A cada iteração (linha 4) são executados 4 passos. No primeiro passo (linha 5), g'' recebe a solução com melhor movimento do conjunto de movimentos resultante de $\Delta(s, g)$. Na linha 6 é excluído

¹É uma técnica de busca populacional que constrói soluções através da combinação de outras soluções para criar novas soluções dentro do espaço de busca.

o melhor movimento do conjunto de movimentos ainda possível, que foi encontrado na linha 5. A solução atual ($\bar{g}$) é atualizada na linha 7. A linha 8 testa se a solução atual $\bar{g}$ é melhor que a melhor solução g' encontrada até o momento. Esta solução g' foi encontrada ao longo da trajetória aplicada a s para chegar a g . Quando a solução corrente chega a g (linha 4) o algoritmo pára e retorna a solução g' (linha 12).

Algoritmo 2.11: *Path Relinking*

```

1  $\bar{g} \leftarrow s$ ;
2  $g' \leftarrow$  a melhor solução entre  $s$  e  $g$ ;
3 Calcular o conjunto de movimentos possíveis  $\Delta(s, g)$ ;
4 enquanto  $|\Delta(s, g)| \neq 0$  faça
5    $g'' \leftarrow$  melhor solução obtida aplicando o melhor movimento de  $\Delta(s, g)$  a  $\bar{g}$ ;
6   Excluir de  $\Delta(s, g)$  o movimento escolhido na linha 5;
7    $\bar{g} \leftarrow g''$ ;
8   se  $(f(\bar{g}) < f(g'))$  então
9     Retorne  $g'$ ;

```

Rosseti [98] ainda dá algumas considerações a respeito da implementação: não aplicar *Path Relinking* a cada iteração e sim periodicamente, para não onerar o tempo final do algoritmo; explorar duas trajetórias potencialmente diferentes, usando s_1 como solução inicial e depois s_2 também; explorar apenas uma trajetória, usando s_1 ou s_2 como solução inicial; não percorrer a trajetória completa de s_1 até s_2 , mas sim apenas parte dela (Reconexão por Caminhos Truncados).

2.3.8 Times Assíncronos

Time Assíncrono ou *A – Team* (do inglês *Asynchronous Team*) foi proposto por Souza e Talukdar em 1992. Foi proposto com base nas relações com sistemas naturais, como sociedade de insetos e comunidades celulares [106]. É uma arquitetura que utiliza de maneira simultânea diversos algoritmos que cooperam entre si, com o objetivo de encontrar soluções ótimas ou quase ótimas para um problema, as quais não seriam encontradas pelos algoritmos isoladamente. Esta seção está baseada em [5, 88, 89, 104, 106].

Também pode ser definido como qualquer organização que consiste de um conjunto de **agentes** (*softwares*), **memórias** e **relações** (entre agentes), sendo que cada um dos **agentes** é responsável por manipular as soluções contidas nas memórias, inserindo, modificando ou apagando as soluções nas memórias. Geralmente, um agente é um algoritmo que se propõe a resolver um problema ou parte dele, com um protocolo de comunicação para compartilharem soluções e outros dados.

As **memórias** são responsáveis por armazenar soluções ou informações relevantes para a resolução do problema, e como elas são compartilhadas, a informação ou resultado produzido por um agente fica disponível aos demais, ou seja, as memórias são uma forma de comunicação entre os agentes. As **relações** definem como as soluções são manipuladas e como os agentes podem acessá-las no conjunto de memórias compartilhadas disponível.

Um *A-Team* pode ser representado graficamente, como mostra a Figura 2.7. Os retângulos equivalem às memórias, as setas os agentes. A memória M_1 é uma composição (memórias compostas por outras memórias básicas), pois contém as memórias M_2 e M_3 . Existem 6 agentes: I , D , A_1 , A_2 , A_3 e A_4 . O agente A_1 lê da memória M_2 e escreve na memória M_3 , A_2 lê de M_4 e escreve em M_2 , A_3 lê de M_1 (ou de M_2 e/ou M_3) e escreve em M_4 e o agente A_4 lê de M_4 e escreve em M_1 (ou em M_2 e/ou M_3). O agente I é o iniciador, que faz o preenchimento inicial das memórias. O agente D é um agente destrutor, que elimina determinadas soluções e controla a quantidade das mesmas.

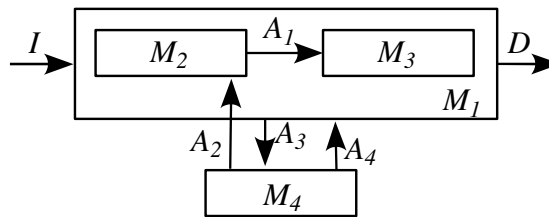


Figura 2.7: Exemplo de *A-Team* com 1 composição, 3 memórias básicas e 6 agentes. Retirado de [5].

A ideia básica dos *A-Teams* é que os agentes trabalhem de forma assíncrona (isto é, de forma paralela), autônoma, interativa e cíclica, sobre um conjunto de dados depositados em memórias compartilhadas fazendo operações de modificar, adicionar, remover ou alterar soluções. O conjunto de agentes forma um time e, como estes agentes são independentes, ou seja, não possuem sincronismo, disponibilizam os resultados de seus trabalhos ou utilizam os resultados de outros componentes do time.

Algumas características baseadas nos princípios das sociedades de insetos podem ser utilizadas de forma a dar eficácia ao projeto do time. Dentre elas, encontram-se:

- Fluxo de Dados Cíclico: as soluções armazenadas nas memórias geradas pelos agentes podem ser disponibilizadas para os outros agentes, de forma de que possam alterar as soluções, com o objetivo de obter uma solução ótima, ou seja, as soluções podem ser refinadas e trabalhadas a todo momento por qualquer agente;
- Autonomia: todos os agentes tem a mesma capacidade de realizar tarefas, ou seja, não há hierarquia entre eles;

- **Comunicação Assíncrona e Paralelismo:** como todos agentes são autônomos, eles executam atividades que podem ser totalmente paralelizadas ou distribuídas. Outro fator favorável ao paralelismo é que, devido à complexidade dos problemas a serem resolvidos, os agentes consomem mais tempo processando dados do que efetuando comunicações com as memórias, portanto a comunicação não é dispendiosa. Não é permitido nenhum tipo de sincronismo durante a execução dos agentes;
- **Consenso Gradual:** inicialmente há uma grande variedade de soluções, mas a partir de certo ponto há um consenso em relação às soluções, restando algumas alternativas disponibilizadas a todos os agentes. Com a cooperação dos agentes, o time em busca de soluções acaba convergindo para um conjunto de soluções possivelmente promissoras.
- **Sinergia:** a cooperação dos agentes pode levar a resultados melhores do que se executá-los separadamente.

Alguns fatores podem influenciar o funcionamento de um *A-Team*, além da cooperação e da manipulação de soluções candidatas. Uma delas é a diversidade de soluções nas memórias. Soluções geradas a partir de soluções em uma memória com pouca diversidade, ou seja, com soluções com poucas diferenças entre si, tem uma forte tendência a ser muito parecidas com as já armazenadas, e há a possibilidade dos agentes ficarem presos em um ótimo local. Mas, se as soluções forem altamente diversificadas, a convergência para uma solução ótima pode ser lenta.

O primeiro passo para modelar um *A-Team* é especificar o problema e suas características, para poder dividir o problema em subproblemas, ou fazer relaxações. Esta é uma tarefa específica e particular de cada problema. Cada decomposição pode ser associada a uma ou mais memórias.

Existem alguns parâmetros de configuração que modelam o time para sua execução. Um dos parâmetros é a **inicialização das memórias**. Este parâmetro é importante, pois pode influenciar fortemente no desempenho geral de um *A-Team* porque está relacionado ao nível de diversidade das soluções. A inicialização com pouco uso do tamanho disponível da memória pode reduzir a diversidade das soluções, mas o preenchimento total pode ser dispendioso, a ponto de soluções geradas serem excluídas antes mesmo de serem processadas. Então, deve-se balancear o uso das memórias de forma a manter uma alta diversidade com uma convergência rápida para uma solução ótima.

Os agentes podem ser divididos em: **agentes de controle**, **agentes iniciadores**, **agentes construtores**, **agentes modificadores**, e os **agentes destrutores**. Um projeto de um time pode conter vários agentes de vários tipos. Os **agentes de controle** servem para manipular os elementos de um *A-Team*. São fundamentais para o projeto e executam atividades como: iniciar e parar o *A-Team*, emitir relatórios sobre os elementos na estrutura ou ativar e desativar elementos.

Os **iniciadores** fazem o preenchimento inicial das memórias, criando novas soluções para o problema em questão. Os **construtores** são responsáveis por produzir novas soluções durante a execução do *A-Team*. Como o objetivo deste agente é o mesmo do que o agente iniciador, eles podem ser agrupados na mesma categoria.

Os **modificadores** têm como função melhorar a qualidade das soluções que estão armazenadas nas memórias. Este agente lê e tenta modificar uma solução, e se conseguir melhorá-la, a substitui no mesmo local onde foi lida.

Os **destrutores** apagam da memória soluções existentes, pois as memórias compartilhadas não podem armazenar soluções indefinidamente. Cada memória está associada a um ou mais agentes destruidores que tem a função de julgar e eliminar, baseados em uma política de destruição, qual solução deve ser eliminada, para dar espaço a uma nova. Geralmente, esta política está relacionada à qualidade dos dados, ou seja, eliminar soluções que não são promissoras.

Os agentes podem ler e escrever nas memórias compartilhadas sempre que desejarem, conforme as **políticas de acesso às memórias**. Cada agente pode gravar sua melhor solução encontrada nas memórias, que podem ser usadas como entrada por outro agente do time. Esta interação entre os agentes resulta em um fluxo cíclico das soluções. Estes ciclos permitem *feedback* e a possibilidade de um agente operar sobre uma solução criada previamente por ele e modificada pelos demais. Esta cooperação é importante pois aumenta as chances do time gerar melhores soluções, que talvez não seriam encontradas com a execução de uma única técnica.

As **políticas de seleção** representam um conjunto de regras a serem consideradas quando é preciso selecionar soluções na memória, para o processamento de qualquer agente. As políticas podem ser implementadas para as memórias ou para os agentes. Algumas políticas de seleção que se destacam são: a **seleção gulosa**, que pega a melhor ou pior solução na memória; a **seleção com distribuição uniforme de probabilidade**, em que todas as soluções têm a mesma probabilidade de serem selecionadas; **seleção com distribuição linear de probabilidade**, em que as soluções possuem uma probabilidade crescente de serem selecionadas.

Como não existem agentes supervisores, cada agente é livre para escolher qual ou quando processar determinada solução. Por isso, eles podem entrar e sair do time a qualquer instante. A saída de um agente causa uma degradação suave no time, não comprometendo todo o time na geração de novas soluções.

A estrutura geral do time pode ser flexível, formado por várias cópias de determinado algoritmo, mas com diferentes parâmetros a serem tratados; formado por vários algoritmos; ou a combinação das duas estruturas anteriores, isto é, diversos algoritmos com várias cópias. Pode-se também utilizar várias memórias em sua estrutura.

A aplicação dos *A-Teams* a diversos problemas mostrou que esta técnica pode ser

muito eficiente na resolução de problemas complexos pertencentes à classe NP-Difícil. Um exemplo é o clássico Problema do Caixeiro Viajante, em que um *A-Team* construído para resolver este problema encontrou soluções ótimas para todas as instâncias testadas [89].

Condução de Experimentos com Heurísticas

O estudo de algoritmos envolve uma etapa essencial, a análise de algoritmos. Os métodos mais utilizados para tal são a análise assintótica e a experimentação. É importante saber qual análise é apropriada no estudo de determinado algoritmo, pois a partir disto pode-se então saber como deve ser conduzida a pesquisa sobre este. Por isso, este capítulo inicia contextualizando em qual destes métodos o estudo de algoritmos heurísticos se encaixam melhor, a experimentação, discutido na Seção 3.1, baseado em [76]. Após, é iniciado o estudo sobre a condução de experimentos com heurísticas, na Seção 3.2.

3.1 Análise de Algoritmos

A partir da análise de algoritmos pode-se estimar o tempo necessário para sua execução, e disto saber se o algoritmo possui um bom desempenho e entender seu comportamento. Esta análise pode ser feita por duas métricas principais: **análise assintótica** e **experimentação**. Na primeira, os algoritmos podem ser analisados por duas diferentes maneiras, por análise de **complexidade de tempo** e por análise de **complexidade de espaço**. Estas análises dão uma estimativa de quanto tempo ou quanta memória um algoritmo pode precisar para encontrar uma solução para determinado problema. Já a experimentação envolve o teste prático do algoritmo projetado, e podem ser feitas medidas de tempo necessário para se encontrar uma solução e de qualidade da mesma.

É de suma importância definir qual método será usado no estudo de um algoritmo, pois a partir desta escolha pode ser definido como será conduzida a pesquisa determinado problema. A seguir, nas Seções 3.1.1 e 3.1.2 são descritos alguns pontos importantes de cada método, para que seja possível compreender como heurísticas podem ser estudadas.

3.1.1 Análise Assintótica

De acordo com Cormen [18], a análise assintótica corresponde ao estudo feito sobre quanto o tempo de execução de um algoritmo ou memória são necessários quando o tamanho da entrada aumenta, ou seja, a análise é feita com base no tamanho da entrada. Um algoritmo que é assintoticamente mais eficiente será a melhor escolha para todas as entradas. Os algoritmos podem ser analisados por duas diferentes maneiras: pelos passos necessários para a resolução de um problema (tais como laços, condições), chamado de **complexidade temporal**, ou por quanta memória de um sistema é necessária para testar um algoritmo, chamada de **complexidade espacial**. Algumas qualidades/vantagens da análise assintótica são:

- Elimina, possivelmente, as dúvidas sobre determinado comportamento, muitas vezes confuso para instâncias pequenas;
- Mostra claramente a taxa de crescimento do tempo de execução;
- Fornece limites claros de tempo e espaço, e também simplifica a análise, eliminando a necessidade de quaisquer suposições (hipóteses) sobre os dados;
- Permite independência de ambiente computacional;
- Facilidade em entender e explicar.

Apesar destas vantagens, paga-se um preço alto por isto, trazendo alguns inconvenientes como:

- O intervalo de valores em que o comportamento assintótico pode ser claramente analisado e exposto pode incluir instâncias de tamanhos que estão bem além de qualquer aplicação concebível no mundo real.
- O pior caso pode ser restrito a um pequeno conjunto de instâncias, não abrangendo todo o conjunto de instâncias encontradas no mundo real. Um clássico exemplo é o método Simplex para Programação Linear. Sabe-se que o comportamento deste método é exponencial no pior caso, mas na prática seu tempo de execução aparece delimitado por um polinômio de baixo grau [2].
- As constantes ocultas na análise assintótica podem dar uma visão distorcida, pois mesmo com taxas de crescimento assintótico razoáveis, a execução de uma implementação pode ser muito custosa, devido à ocultação destas constantes.
- Mesmo na ausência de quaisquer dos problemas acima citados, encontrar bons limites para muitos problemas pode ser complicado. Muitos dos principais algoritmos de aproximação para problemas NP-Difíceis possuem esse inconveniente: considerando um grande número de parâmetros e os avanços desenvolvidos sobre o método, criam um ambiente complexo que é muito difícil de analisar com métodos existentes, no que diz respeito ao tempo de execução e à qualidade da solução encontrada.

- A análise assintótica tende a favorecer o desenvolvimento de algoritmos chamados de “*paper-and-pencil*”, ou seja, algoritmos que nunca são implementados. Algoritmos deste tipo ignoram frequentemente técnicas importantes para fazer implementações eficientes [76].

3.1.2 Experimentação

Segundo Moret [76], a implementação sempre foi utilizada no trabalho de desenvolvimento e análise de algoritmos e estruturas de dados. Essa é uma parte da pesquisa de algoritmos, chamada de **experimentação algorítmica**, que pode ser entendida como uma metodologia de pesquisa no desenvolvimento de algoritmos e estruturas de dados. A abordagem baseada em **experimentação** é chamada de **experimental** ou **empírica**, que significa que o experimento é baseado em experiências, observações e tentativas, que podem ser confirmadas ou rejeitadas através dos experimentos.

A abordagem empírica é bem aplicada nas Ciências Naturais, pois é da natureza que se encontram os resultados empíricos. É chamada de **método científico**, e tem sido utilizada durante séculos. Mas essa abordagem pode não ser suficiente no mundo matemático, artificial. Nas ciências naturais não existe outra maneira a não ser aprender utilizando ou observando a natureza, e os modelos construídos se baseiam em modelos com medidas retiradas da natureza. Já para aprender sobre algoritmos, não há um método, o que é medido são os resultados que o algoritmo encontra, e os resultados são simplesmente relatados e comparados com outros experimentos feitos da mesma maneira.

Moret [76], Coffin e Saltzman [15] citam uma possibilidade para contornar as situações citadas no uso da análise assintótica: a **experimentação**. Sem abandonar o uso da análise assintótica (geralmente um estudo sobre o pior caso), mas sim complementando o estudo sobre determinado problema com a experimentação, o que implica que os algoritmos devem ser implementados, não apenas projetados.

Mas, o que é um experimento? Montgomery [75] define **experimento** como um ensaio ou uma série de ensaios nos quais são feitas mudanças nas variáveis de entrada de um processo ou sistema, com o objetivo de identificar e observar as razões para as mudanças na resposta de saída. Um experimento consiste de um conjunto de procedimentos realizados com o objetivo de levantar evidências sobre a validade de uma hipótese levantada, isto é, execução de um conjunto de testes com um objetivo específico. Para demonstrar um fato conhecido, checka-se a validade da hipótese, que pode ser baseada tanto em algoritmos já conhecidos como em novos algoritmos. Pesquisadores de diversas áreas de estudo efetuam experimentos para demonstrar uma teoria, para descobrir conhecimento sobre um processo particular, e medir o efeito de um ou mais fatores sobre algum fenômeno.

Em relação a testes computacionais de um algoritmo, um experimento consiste em resolver uma série de instâncias de um problema usando uma implementação do algoritmo. O pesquisador deve implementá-lo, selecionar as instâncias, escolher um ambiente computacional, escolher as medidas de desempenho, configurar as opções do algoritmo, e finalmente relatar os resultados. A escolha feita para cada um destes fatores pode ter um efeito substancial sobre os resultados e a relevância do experimento.

Algumas questões em relação ao desempenho de um algoritmo ou em relação à classe de problemas a se trabalhar são fáceis de responder, até mesmo antes de implementar um algoritmo, mas outras só poderão ser respondidas com a execução dos experimentos. Diante disso, Barr et al. [6] citam alguns quesitos desejáveis para métodos heurísticos, ou seja, um novo método heurístico traz contribuições quando é:

- **Rápido:** produz soluções de alta qualidade mais rápido que outras abordagens;
- **Preciso:** identifica soluções de alta qualidade mais rápido que outras abordagens (Ahuja e Orlin [3]);
- **Confiável:** provê soluções viáveis e corretas (Ahuja e Orlin [3]);
- **Robusto:** menos sensível a diferenças nas características do problema, qualidade dos dados, refinamento de parâmetros melhor que outras abordagens (Ahuja e Orlin [3], Hopfield e Tank [55]);
- **Simples:** fácil de implementar (Ahuja e Orlin [3], Dyer e Frieze [30], Lin e Kernighan [61]);
- **De alto impacto:** resolve um problema novo e importante mais rápido e preciso que outras abordagens (Rothfarb et al. [99]);
- **Generalizável:** tem aplicação para um grande número de problemas (Feo e Resende [31], Glover [38], Holland [52], Metropolis et al. [70]);
- **Inovador:** possui ideias novas e criativas;
- **Revelador:** oferece compreensão no projeto geral da heurística ou da estrutura do problema, estabelecendo as razões para esse desempenho e explica este comportamento;
- **Teórico:** provê compreensões teóricas, como limites sobre a qualidade da solução (Held e Karp [47, 48], Hochbaum e Shmoys [51], Johnson e Papadimitriou [56]).

Como foi visto que a experimentação é essencial no estudo de algoritmos heurísticos, e diante dos quesitos desejáveis vistos, a Seção 3.2 inicia o estudo sobre os principais passos encontrados na literatura, a serem seguidos na condução de experimentos com heurísticas.

3.2 Passos para Condução de Experimentos Utilizando Heurísticas

De fato, o primeiro passo a ser dado para execução de um experimento é a revisão da literatura. Johnson [57] afirma que um fator chave para publicar um artigo é contextualizá-lo em relação ao estado da arte. Além disso, Barr et al. [6] elaboram, de maneira simples e objetiva, mais cinco passos a serem seguidos para realizar experimentos com algoritmos, e estão organizados da seguinte maneira:

1. Fazer uma revisão da literatura;
2. Definir os objetivos do experimento;
3. Escolher medidas de desempenho e fatores a explorar;
4. Projetar e executar o experimento;
5. Analisar os dados e mostrar as conclusões; e
6. Relatar os resultados dos experimentos.

Todos os passos serão descritos: passos 1 a 3 nas seções 3.3, 3.4, 3.5, passo 4 no Capítulo 4 e para concluir, os passos 5 e 6 no Capítulo 5, respectivamente. Para cada passo serão detalhadas algumas abordagens recomendadas, de acordo com [6, 19, 57, 67, 76, 95], dentre outros autores.

3.3 Revisão da Literatura

De acordo com Bisquerra et al. [8] a revisão da literatura fornece: um marco de referência conceitual; a compreensão do estado da questão; indicações e sugestões quanto ao enfoque, ao método e a instrumentação para análise de dados; uma estimativa das possibilidades de êxito, da significação e da utilidade dos resultados; informação específica necessária para elaboração de definições, suposições, limitações e, basicamente, das hipóteses.

Segundo McGeoch e Moret [68, 76], deve-se prover o contexto da pesquisa, ou seja, deve-se saber se o algoritmo já foi estudado, se já foi implementado e testado, o que já foi estudado e feito sobre o problema. Se o problema é novo, o que se espera alcançar com a abordagem experimental.

Quando se trabalha com a abordagem experimental, uma parte crucial da descrição do problema é a motivação para conduzir o experimento. Deve-se explicar porque uma análise analítica não é suficiente para o problema. Seria devido às propriedades dos dados? Ou a natureza complexa do problema (comum em muitos problemas de otimização)? Como nem todos os problemas são abordados experimentalmente, essa escolha deve ser justificada.

Definido um problema, existem fatores a serem considerados em relação a este, como por exemplo, as modelagens existentes, métodos desenvolvidos, problemas relacionados, se o problema em estudo possui limites inferiores conhecidos, se existem instâncias disponíveis para teste, ou geradores de instâncias existentes.

3.4 Objetivos do Experimento

Para Barr et al. e Moret [6, 76], um experimento de uma pesquisa deve ter um objetivo especificado claramente, deve ser o ponto de partida da pesquisa. A partir dele é que serão respondidas as questões, na qual a experimentação é necessária. É nesta fase que o pesquisador lista as hipóteses a testar, os resultados a procurar e quais fatores explorar.

Definido um problema a estudar, Kerlinger, 1981 apud Bisquerra et al. [8] citam que para que um problema possa ser objeto de estudo científico, deve satisfazer no mínimo três condições:

1. Deve expressar uma relação entre duas ou mais variáveis;
2. A formulação deve ser clara, sem ambiguidades, e se possível, em forma de pergunta; e
3. Deve permitir uma verificação empírica.

Geralmente, um dos objetivos do experimento é comparar uma nova abordagem com técnicas já implementadas. Para isso, Johnson [57] recomenda que se obtenha o código da implementação de um algoritmo anteriormente utilizado e relate os resultados utilizando seu ambiente computacional (para verificar também se os resultados são consistentes). Se isto não é possível, outra opção é desenvolver uma implementação comparável do algoritmo, para verificar a consistência dos dados. Se os experimentos anteriores foram realizados em uma arquitetura desconhecida, isto pode ser impossível. Entretanto, é possível fornecer algumas estimativas grosseiras, por exemplo, se a implementação é mais rápida que outras.

Para Moret [76], um erro que não deve-se cometer é comparar duas linguagens ou plataformas diferentes, ou mesmo comparar dois algoritmos que têm comportamentos totalmente distintos (por exemplo, linear ou quadrático).

Em geral, os algoritmos podem ser testados novamente, incluindo melhorias. Se heurísticas bem conhecidas possuem instâncias de referência, uma nova heurística pode ser testada e comparada com esses valores. Além disso, podem ser feitas comparações com os resultados publicados sobre diferentes problemas e máquinas. Se não existem outros métodos para comparação, então um método mais geral como programação linear ou inteira, ou uma simples abordagem gulosa pode servir como base. Algumas heurísticas

baseadas em elementos probabilísticos como GRASP¹, Algoritmos Genéticos², Busca Tabu³, podem ser utilizadas para comparação.

Especificamente, Moret [76] descreve alguns estudos que podem ser feitos na pesquisa empírica de algoritmos:

- Verificar a corretude e precisão em casos extremos;
- Medir o tempo de execução de programas baseados em algoritmos exatos com instâncias reais de problemas NP-Difíceis;
- Avaliar a qualidade das heurísticas para soluções aproximadas de problemas NP-Difíceis (podendo gerar instâncias difíceis);
- Comparar o desempenho atual de algoritmos para problemas tratáveis;
- Descobrir os *speed-ups*⁴ alcançados por algoritmos paralelos;
- Investigar e refinar critérios e fatores de otimização direcionado ao uso real;
- Testar a qualidade e robustez das simulações, e estratégias de otimização para sistemas complexos etc.

Johnson [57] sugere que se faça uma reflexão antes de começar a trabalhar. Algumas questões que podem ser elaboradas para serem respondidas são:

- Que fenômeno algorítmico você quer estudar?
- Quais as questões que você quer levantar com seus experimentos?
- Qual comportamento do algoritmo precisa ser explicado?
- Qual (quais) parte(s) do algoritmo precisa(m) ser melhorada(s)?
- Alguém além de você já respondeu as questões levantadas por você, dada a atual situação da literatura na área?
- Você implementou o algoritmo corretamente, incorporando as características/variações que você quer estudar e fornecer a produção de todos os dados de saída necessários?
- O conjunto de instâncias a ser testado é adequado para responder as questões levantadas?
- Quais tipos de instâncias de testes não foram estudadas adequadamente?
- Dada a velocidade dos computadores atuais e capacidade de memória, quais instâncias produzem diferenças significativas em relação à qualidade da solução, para alcançar tempos de execução viáveis?

Moret [76] cita uma lista de possíveis objetivos de pesquisa:

¹Para mais detalhes da técnica GRASP, ver Seção 2.3.4.

²Para mais detalhes sobre Algoritmos Genéticos, ver Seção 2.3.5.

³Para mais detalhes da técnica Busca Tabu, ver Seção 2.3.3.

⁴*Speed-up* é uma medida de desempenho que compara o tempo de execução entre algoritmos sequenciais e paralelos. Pode ser calculada pela razão de Desempenho sequencial por Desempenho paralelo.

- **Testar e melhorar algoritmos para problemas difíceis:** Entender como uma heurística trabalha para diminuir o tempo computacional ou delimitar a qualidade das aproximações obtidas. Estes aspectos são fundamentais na avaliação do desempenho para auxiliar a produzir heurísticas melhores.
- **Comparar algoritmos existentes e estruturas de dados para problemas:** Fazer experimentos facilita a identificação de implementações boas ou ruins, e se a melhoria obtida pela teoria também é válida na prática. Novas conclusões podem ser inferidas para contribuir para um refinamento ou simplificação de um algoritmo.
- **Comprovar e Refinar Conjecturas:** Testar conjecturas sobre uma série de casos pode, no mínimo, evitar fazer um trabalho que poderá ser desperdiçado futuramente. Bons experimentos são uma fonte rica para novas conjecturas e teoremas.
- **Desenvolver bibliotecas para algoritmos básicos e estruturas de dados:** Deve-se implementar algoritmos que garantam que o tempo de execução seja eficiente e deve-se documentar os casos em que ele tem um desempenho bom ou ruim.
- **Desenvolver ferramentas para facilitar o projeto e análise de algoritmos:** Nesta categoria se enquadram ferramentas gráficas e estatísticas para analisar experimentos, podendo conter ferramentas de animação para visualizar o progresso de um experimento. Estas ferramentas podem ilustrar uma grande quantidade de informação de maneira fácil de entender.

3.5 Medidas de Desempenho e Fatores a Explorar

Em um experimento computacional existem vários elementos que podem ser analisados, como por exemplo, os conjuntos de variáveis dependentes e independentes⁵. Todos estes elementos devem ser analisados de acordo com as metas do experimento.

Segundo Rardin e Uszoy [95], medidas de desempenho são necessárias, principalmente na fase de desenvolvimento da heurística, quando os algoritmos básicos são projetados, para futuramente obter uma implementação eficaz, que trabalhará com uma configuração específica.

Podem ser encontrados na literatura [2, 3, 6, 66, 76, 95] uma variedade de critérios para avaliar métodos heurísticos e utilizar medidas de desempenho. Geralmente, as medidas de desempenho são divididas em três áreas: **qualidade da solução**, **esforço computacional** e **robustez** [6]. Entretanto, Rardin e Uszoy [95] consideram que a principal medida de desempenho nos experimentos com heurísticas é o tempo, que está inclusa nos esforços computacionais.

⁵As variáveis independentes são aquelas que são manipuladas pelo pesquisador. As variáveis dependentes são consequência das variáveis independentes, e são apenas medidas ou registradas. Para mais detalhes ver Seção A.2, do Apêndice A.

Qualidade da Solução

Dois fatores são importantes quando um algoritmo testado encontra uma solução ótima para o problema dado: velocidade e taxa de convergência (ou seja, em quantos passos o algoritmo converge para uma solução ótima). Para heurísticas, uma consideração adicional é de como a otimalidade da solução é tratada pela heurística, ou seja, como é possível saber se o algoritmo está gerando boas soluções, pois sabe-se que heurísticas não garantem encontrar soluções ótimas.

Uma medida que pode mostrar uma estimativa sobre a qualidade da solução é a **acurácia**, que é a diferença entre o valor encontrado e o valor de referência. O valor de referência pode ser uma solução encontrada por meio de métodos exatos, ou pode ser o melhor valor encontrado por alguma heurística [60].

Realmente é um grande desafio avaliar a qualidade das soluções encontradas por heurísticas, pois geralmente os problemas resolvidos com heurísticas são NP-Difíceis, e muitas vezes para determinados problemas, métodos exatos não produzem soluções em tempo viável. Infelizmente, ainda não existe um método satisfatório para fazer a análise da qualidade de soluções geradas por heurísticas. Para utilizar a qualidade da solução como medida de desempenho, existem alguns métodos que são empregados, tais como: calcular a solução exata para pequenas instâncias; utilizar limites inferiores ou superiores; construir instâncias a partir de valores ótimos conhecidos; aplicar estimativa estatística de valores ótimos conhecidos; e comparar os melhores valores encontrados [95].

Na primeira abordagem, soluções ótimas exatas são encontradas para pequenas instâncias, e então essas soluções são comparadas com as obtidas pelas heurísticas. Depois, a heurística é aplicada às instâncias maiores e de interesse prático, e as medidas de desempenho das instâncias resolvidas são utilizadas para comparação. A ideia é poder tornar representativos os dados que foram fáceis de obter.

Porém, esta abordagem pode ser um tanto duvidosa, pois o comportamento de heurísticas para pequenas instâncias pode ser totalmente diferente para grandes instâncias, e quanto maior for a instância, pior pode ser o comportamento da heurística. Outro problema que pode ocorrer é que se os parâmetros estiverem mal configurados ou se decisões na parte de construção da solução forem tomadas erroneamente, a heurística pode ter um desempenho péssimo com instâncias grandes, ou encontrar soluções muito longe de soluções ótimas para pequenas instâncias. Então, como podem ocorrer erros grosseiros ao avaliar algoritmos a partir de instâncias pequenas, o tempo e esforço gastos para desenvolver o método exato podem ser gastos na melhoria de uma heurística ou no desenvolvimento de outras.

Na segunda abordagem, são calculados limites para o valor ótimo e comparados com a solução produzida pela heurística. A vantagem desta abordagem é a possibilidade

de se calcular o desvio da otimalidade⁶ da solução heurística. No entanto, existem algumas desvantagens. Se o limite superior é folgado, não fica claro se o desvio encontrado em relação ao ótimo é devido ao desempenho pobre da heurística ou se o próprio limite está muito longe do valor ótimo. Além disso, calcular limites apertados é uma tarefa difícil e complexa, muitas vezes NP-Difícil. Uma parte considerável do experimento é gasto para calcular limites, com intuito de avaliar com mais precisão o desempenho heurístico.

Uma opção seria combinar as duas abordagens citadas anteriormente, ou seja, comparar soluções ótimas para pequenas instâncias, comparar os limites, e adaptar as medidas de erro dos limites para instâncias grandes. Mas esta abordagem tem as mesmas desvantagens de generalizar o desempenho da heurística com resultados encontrados em instâncias pequenas.

A abordagem de comparar os melhores valores encontrados é uma técnica muito utilizada. Em alguns casos, quando existem valores publicados em bibliotecas de referência, a melhor solução encontrada vem de outro pesquisador. Quando são feitos muitos testes e chega-se à mesma faixa de valores, pode-se assumir que a melhor solução conhecida é ótima ou próxima da ótima.

Quando não são conhecidas soluções, uma maneira de encontrar melhores soluções é fazer várias execuções nos algoritmos. Um algoritmo de Busca Local pode ser executado várias vezes com diferentes soluções iniciais para obter uma boa aproximação para o valor ótimo, ou aplicação de outro método, como Busca Tabu, e executá-lo depois do critério de parada que foi implementado para os experimentos.

Outra forma de encontrar uma melhor solução é registrar os melhores valores encontrados por qualquer um dos algoritmos do experimento. Quando vários algoritmos são utilizados para testar o mesmo conjunto de instâncias de teste, especialmente quando os algoritmos seguem estratégias diferentes, os valores encontrados podem estar próximos de valores ótimos.

Esforço Computacional

Em relação ao **esforço computacional**, a velocidade de computação é um fator chave. Várias partes do processo podem ser cronometradas, como [6]:

Tempo da melhor solução encontrada: Esse é o tempo necessário para a heurística encontrar uma solução. Através dele é possível identificar se a heurística é rápida. Pode envolver todos os processos, juntamente com todos os pré-processamentos [19, 20].

⁶O desvio de otimalidade neste contexto, é a distância percentual entre solução encontrada e solução ótima.

Tempo médio total de execução: É o tempo médio total dos tempos de processamento necessários para executar o algoritmo, ou seja, a média dos tempos de todos os testes feitos sobre uma mesma configuração e mesma instância.

Tempo por fase: Quando uma heurística é multi-fase ou composta, como por exemplo, solução inicial, melhor solução e solução final, o tempo de cada fase e a qualidade da solução até o final de cada fase também pode ser relatada. Essas medidas mostram em qual fase a heurística é mais dispendiosa.

Uma outra medida de tempo que pode ser obtida é o tempo que a heurística gasta para convergir para uma solução ótima. A relação entre esforço computacional e qualidade de uma solução pode ser calculada pela equação 3-1, a qual fornece a relação entre o tempo necessário pela heurística para encontrar uma solução de valor a 5% do valor da melhor solução encontrada (t^*) e essa [6]:

$$r_{0.05} = \frac{t_{0.05}}{t^*} \quad (3-1)$$

Este cálculo pode ser muito útil, entretanto, os tempos de execução não podem ser comparados de um sistema computacional para outro. Logo, quando forem aplicados esses cálculos e comparados, os testes devem ser feitos no mesmo ambiente computacional.

Johnson [57] lembra que mesmo que o objetivo principal não seja analisar os tempos de execução, eles devem ser relatados, porque o leitor pode querer saber essa informação antes de fazer um estudo detalhado dos resultados. Por exemplo, se o principal objetivo são os cálculos combinatórios e operações algorítmicas, o leitor pode querer fazer uma correlação das operações com o tempo de execução. Algumas avaliações de algoritmos enumerativos, baseados em árvore de busca, que contêm estratégias de poda (dado um limite inferior ou superior, subárvores são eliminadas da busca) medem somente a quantidade de subproblemas explorados (nós da árvore de busca). Isso pode levar a conclusões erradas, porque o algoritmo pode encontrar uma solução com poucos nós, mas se cada nó demora um tempo razoável para ser computado, pode ser preferível utilizar um algoritmo que calcula vários nós mas que leva um tempo menor de computação global. Se o objetivo é medir a qualidade da solução, é importante saber que um dos motivos de trabalhar com algoritmos de aproximação é dar um resultado com qualidade, mas que execute em um tempo reduzido, por isso é importante relatar o tempo de execução. Em resumo, o autor que quer reproduzir os resultados de algum trabalho sempre está interessado no tempo de computação envolvido.

Entretanto, deve-se analisar o tempo que será relatado. Johnson [57] afirma que muitos artigos relatam um tempo de execução de um segundo ou menos. É claro que um algoritmo que leva 0,01 segundo tem uma vantagem sobre um que executa em 0,1

segundo, porque é 10 vezes mais rápido, mas na maioria das aplicações, se o algoritmo gasta um segundo ou menos, isso é irrelevante. O diferencial pode ser visto se há a necessidade de resolver rapidamente milhares de casos, ou se este fator de 10 faz diferença para instâncias maiores, mas elas devem ser testadas para garantir esta vantagem.

Robustez

Uma heurística que obtém uma solução excelente para apenas uma instância do problema não é robusta e também não é interessante. Em geral, a robustez é baseada na capacidade da heurística de encontrar soluções para uma grande variedade de instâncias e é mostrada com as **medidas de variância** [6, 19, 43].

A robustez pode ser demonstrada apresentando o estudo de uma instância do problema. Quando os valores dos parâmetros são escolhidos, algumas medidas de sensibilidade da heurística podem ser notadas modificando alguns desses parâmetros. Deve-se também relatar os resultados negativos. Por exemplo, pode ocorrer da heurística resolver várias instâncias, mas falhar em algum ponto específico. Os testes devem ser feitos para todas as instâncias.

O **teste de stress** também é muito útil. Seu objetivo é realizar testes com a maior quantidade de instâncias possível e depois analisar quais fatores influenciam no desempenho do algoritmo.

Geralmente, os leitores de artigos ou relatórios muitas vezes não estão interessados somente nos resultados finais, mas também nas contribuições encontradas com as respectivas estratégias utilizadas. Assim, deve-se relatar as estratégias utilizadas, configurações de controle, seleção de heurísticas, códigos e novas ideias que podem ser usadas em outros contextos.

Quando o desempenho do algoritmo está fortemente ligado à estratégia e aos parâmetros de escolha, a pesquisa gastará muito esforço escolhendo as opções apropriadas e as configurações dos parâmetros. O processo utilizado para fazer essas escolhas é de interesse do leitor, e elas devem ser relatadas. Outro fator importante é que uma heurística robusta que execute bem sobre vários parâmetros é superior às heurísticas que requerem configurações únicas para cada instância do problema, a menos que a heurística seja projetada para se auto-ajustar baseada nas características do problema.

Outras Medidas

Além dos fatores qualidade da solução, esforços computacionais e robustez, Crowder et al. [19, 20] apontam outros indicadores de desempenho. Os principais são: **tradeoffs** (neste trabalho, um *tradeoff* é uma comparação feita entre dois fatores, onde é verificado qual fator perde ou ganha na execução de um experimento. Tenta-se achar

um equilíbrio para os dois fatores, com o objetivo de ter um bom desempenho nos testes); a **precisão numérica** (uma medida de capacidade do algoritmo de computar corretamente a resposta diante da instabilidade numérica); a **quantidade de iterações** (para algoritmos iterativos, a quantidade de passos que o algoritmo necessita para resolver o problema, sendo que este indicador é independente do computador usado); **quantidade de chamadas de uma determinada função** (esta função pode ser chamada da própria função objetivo ou de outra função que tenha relevância no teste); **operações matemáticas** (a quantidade de vezes que uma operação básica é necessária durante a execução do algoritmo); **armazenamento dos requisitos e estruturas de dados** (pois quando a instância aumenta, a necessidade de armazenamento de dados também aumenta); **interface amigável ao usuário** (fácil de usar e portátil).

Fatores como **quantidade de nós gerados numa árvore de busca**, **altura da árvore**, **tempo de execução por nó** podem ser relatados. Dessa forma, pode ficar mais fácil a comparação, visto que essas medidas são independentes de máquina [54].

Projeto e Execução do Experimento

Vistos os três passos iniciais na condução de experimentos, a revisão da literatura, definição dos objetivos do experimento, escolha de medidas de desempenho e fatores a explorar, encontrados no Capítulo 3, o próximo passo é o projeto e a execução do experimento. A escolha de como conduzir o experimento é um dos principais passos a serem feitos em experimentação. Na Seção 4.1 é definido o que é Planejamento Experimental, e citados alguns modelos experimentais encontrados na literatura e que mais se adequam à experimentação algorítmica.

Outro passo durante o projeto do experimento é trabalhar com instâncias para testar os algoritmos desenvolvidos. Dependendo do problema, há a necessidade de criá-las. A Seção 4.2 aborda sobre questão.

Outras questões são tratadas neste capítulo tais como a escolha de critérios de parada (Seção 4.3), execução dos testes (Seção 4.4), ajustes de parâmetros (Seção 4.5), algumas questões de implementação (Seção 4.6), e finalmente tempo gasto na execução do experimento (Seção 4.7).

Para leitura deste Capítulo é recomendável ler o Apêndice A, que mostra alguns conceitos básicos de Estatística.

4.1 Planejamento Experimental

Um bom experimento tem por objetivo alcançar as metas experimentais, demonstrar claramente o desempenho dos testes, ter justificativas lógicas, gerar boas conclusões e ser passível de reprodução. Todas estas características têm um valor importante na experimentação com métodos heurísticos.

O **modelo experimental**, que equivale a um conjunto de parâmetros considerados para analisar os métodos desenvolvidos para solucionar um dado problema, deve ser definido cuidadosamente, para que se possa fazer inferências sobre o desempenho do algoritmo que será implementado. Produzir um modelo experimental de confiança envolve: identificar as variáveis que podem influenciar no desempenho; decidir as medidas apropriadas de desempenho e avaliar a variância destas medidas, selecionando um conjunto

apropriado de instâncias de teste para poder responder às questões que são levantadas nos objetivos do experimento [19, 20].

Para alcançar os objetivos de um experimento, Rardin e Uzsoy [95] citam quatro tipos de modelos experimentais, a saber: **Básico** (instâncias $\times$ algoritmos), **Modelo Experimental Refinado**, (chamado de Planejamento Estatístico de Experimentos ou *Statistical Design of Experiments*), **Blocagem de Instâncias** e **Balanceamento de Qualidade e Tempo**.

O primeiro modelo, o Básico, corresponde a um planejamento experimental simples, em que os dados são armazenados em uma tabela, onde as linhas correspondem às instâncias do problema, e as colunas aos algoritmos que foram testados, como na Figura 4.1. Cada célula da tabela equivale a um tempo de execução de um algoritmo sobre uma instância. Os algoritmos podem ser diferentes, mas geralmente são variações de uma mesma ideia. As instâncias podem ser completamente diferentes e independentes, mas geralmente são organizadas por características similares, como o tamanho.

	Algoritmo 1	Algoritmo 2	...	Algoritmo n
Instância 1				
Instância 2				
...				
Instância m				

Figura 4.1: Abordagem básica: instâncias $\times$ algoritmos.

Para construir um Modelo Experimental Refinado, deve-se iniciar com um conjunto de questões sobre as heurísticas em estudo que precisam ser respondidas. Tais questões, em geral dizem respeito a como as diferentes características do problema (tais como tamanho do problema, quantidade e natureza das restrições) e parâmetros dos algoritmos (critério de parada, busca na vizinhança e seleção de movimentos) afetam o desempenho das heurísticas sendo testadas.

Definidos os fatores e seus respectivos níveis¹, este modelo consiste em identificar todas as características do problema e dos algoritmos, que podem influenciar nos resultados, executando todas as combinações entre os níveis dos fatores, e ao final, avaliar os resultados para ver o que pode ser concluído.

Geralmente, os fatores das instâncias e dos algoritmos são testados variando-se dois níveis. Isto permite analisar se o fator influi realmente no resultado. Mas deve-se saber quais níveis serão escolhidos, pois caso contrário, podem dar resultados errados. Pode-se testar vários níveis por fator. Por exemplo, suponha um algoritmo genético, que

¹ A definição de fator e nível é encontrada na Seção A.2.

contenha 8 fatores a serem analisados. Se cada fator possuir somente dois níveis, obtêm-se $2^8 = 256$ combinações. Fazer três réplicas dessas combinações dá $256 \times 3 = 768$ casos a serem analisados.

Para minimizar a quantidade de testes e maximizar a informação adquirida, um processo simples é feito para analisar e estruturar experimentos, garantindo que os dados coletados possam ser analisados por modelos estatísticos para obter conclusões válidas e objetivas. O modelo experimental refinado é baseado nos princípios de replicação (testes repetitivos) e randomização (realizar testes de forma aleatória para compensar fatores não incluídos). Existem alguns métodos tais como a Análise de Variância de Um Fator (*Single-factor Analysis of Variance - ANOVA*)², o Fatorial Completo³ ou Quadrado Latino⁴, que são muito utilizados para análise estatística de experimentos [75]. Quando os resultados de um experimento variam de acordo com as condições de teste (parâmetros de configuração, ambiente computacional, problemas resolvidos), a metodologia estatística é a única abordagem objetiva de análise. Por isso, o projeto de um experimento e análise estatística dos dados são inter-relacionados [95].

Após o estudo sobre um problema, deve-se decidir quais algoritmos serão utilizados na resolução do mesmo, e também quais variações do algoritmo serão aplicadas. A estratégia deve ser explicada, e por que o algoritmo foi escolhido. O teste-piloto pode ajudar na escolha das estratégias.

O teste-piloto é o principal instrumento para decidir quantos e quais são os níveis que têm significado prático. Desta forma, pode-se investigar e descobrir quais níveis de determinado fator são realmente necessários, ou seja, permitem descobrir quais fatores e níveis que podem ser refinados. As regras de parada geralmente são encontradas a partir de configurações iniciais destes testes.

Portanto, o teste-piloto pode dar uma ideia de quanta variabilidade pode ser esperada nos resultados, ou seja, quantas replicações e repetição das execuções de algoritmos aleatórios são necessárias para obter resultados confiáveis. Contudo, cuidados devem ser tomados ao configurar um algoritmo para trabalhar bem em um conjunto específico de instâncias, pois os resultados podem não ser gerais para a população total de instâncias. Uma das vantagens de fazer testes preliminares é que o experimento não fica comprometido, com parâmetros mal configurados.

Projetos experimentais sequenciais ou múltiplos podem ser muito eficientes. Por exemplo, uma primeira rodada de experimentos poderia incluir todos os fatores do algoritmo em dois níveis com pouca ou nenhuma replicação. Com a análise destes

²O método ANOVA é descrito em detalhes na seção A.4.1, no Apêndice A.

³O método Fatorial Completo é descrito na seção A.4.2, no Apêndice A.

⁴O método Quadrado Latino é descrito na seção A.4.3, no Apêndice A.

primeiros resultados, pode-se inferir quais fatores que podem ter os níveis fixados, e focar na replicação sobre os algoritmos e fatores principais da pesquisa.

O Planejamento Fatorial Completo, no caso de experimentos computacionais, pode ser aplicado executando todos os algoritmos sobre todo o conjunto de instâncias de teste. Normalmente, pode ser viável aplicar esse tipo de planejamento, pois o custo de realizar experimentos computacionais é baixo, mas somente se forem poucos os fatores a serem investigados.

Um maneira de não executar todas as combinações é utilizar o Planejamento Fatorial Fracionário. Para utilizar este método, deve-se escolher cuidadosamente os fatores e níveis a serem estudados. Quando nenhuma outra técnica é encontrada para diminuir o tamanho do experimento, essa técnica pode ser útil. Entretanto, como experimentos com heurísticas requerem replicação, o método fatorial fracionário não é apropriado.

Outra abordagem, a Blocação de Instâncias (*Blocking of Instances*), tem seus princípios baseados na técnica de randomização, que no planejamento estatístico de experimentos define que todos os fatores não controláveis do experimento devem ser definidos de forma aleatória. A aplicação desse princípio deve ser rigorosa, pois instâncias aleatórias e diferentes devem ser escolhidas para cada algoritmo.

Apesar deste princípio de randomização, a blocagem de instâncias é feita testando todos os algoritmos sobre o mesmo conjunto de instâncias de teste. Lin e Rardin, 1979 apud [95], demonstram que as diferenças entre algoritmos são mais fáceis de serem encontradas estatisticamente se os mesmos conjuntos de instâncias forem testados por todos os algoritmos.

Por último, o experimento pode ser conduzido aplicando o Balanceamento de Qualidade e Tempo. Geralmente, uma das principais características em experimentos computacionais é a necessidade de descrever os *tradeoffs*⁵ entre tempo para obtenção de uma solução e quão perto ela é de uma solução ótima. É necessário criatividade para tratar de alguns *tradeoffs*. O princípio dessa abordagem é que todos os algoritmos consumam a mesma quantidade de recursos computacionais. Isso é muitas vezes a maneira mais eficaz para comparar algoritmos heurísticos bastante diferentes.

Considere, por exemplo, um experimento comparando um algoritmo de Busca Local simples com um algoritmo de Busca Tabu. Os dois têm a mesma estrutura de busca local, mas o algoritmo de Busca Tabu permite vários movimentos sem melhoria. Com isso, é possível obter melhores soluções, mas o tempo de execução pode ser bem maior.

Para ter um experimento mais justo, o algoritmo mais simples pode ser executado mais vezes, ou seja, repetir a execução, com diferentes inicializações aleatórias, uma certa quantidade de vezes ou até alcançar algum valor na função objetivo. Nessas circuns-

⁵Ver Seção 3.5, em Outras Medidas.

tâncias, o experimento pode ser mais justo, permitindo que o método mais simples seja executado mais de uma vez. Então o algoritmo de busca tabu é executado com os mesmos limites, e comparado.

Enfim, ao utilizar análise estatística em experimentos, essa requer um novo conjunto de normas para a investigação. Modelos experimentais são muito empregados em Física, Engenharia e Medicina. Padrões para testes empíricos em Computação e Matemática têm sido menos rigorosos e por isso há uma ampla gama de métodos aceitos para a análise de dados que não utilizam todo o rigor estatístico.

No entanto, os resultados experimentais devem ser avaliados para contribuir para o entendimento do problema e dos algoritmos em estudo, ao invés de mostrar simplesmente que um algoritmo é melhor que outro. Portanto, as técnicas de avaliação de algoritmos utilizando conceitos estatísticos são uma opção para delinear um experimento computacional.

Alguns trabalhos que utilizam métodos estatísticos para análise de algoritmos os de McGeoch [66], o qual analisa o problema *Self-organization Search*, com o uso de técnicas de redução de variância e o trabalho de Rardin e Uzsoy [95], que utilizam ANOVA para analisar algoritmos como Busca Tabu e *Multistart* para o problema de *One Machine Scheduling*.

4.2 Seleção do Conjunto de Instâncias de Teste

Uma das maiores tarefas na condução de experimentos computacionais é construir as instâncias de teste. Não importa o quão bem feita é a estrutura do projeto experimental, se não houver disponibilidade de dados suficientes, reais e com variedade que abranja todas as características do problema em questão [95].

Em geral existem dois tipos de instâncias para serem testadas: aquelas que representam aplicações de problemas do mundo real e aquelas que são instâncias geradas aleatoriamente. Ambas têm sido usadas na validação de programas e testes. As instâncias construídas podem ser modeladas como uma maneira de testar aspectos específicos que muitas vezes os problemas reais não abordam. Além do mais, instâncias geradas podem ser usados para testar procedimentos e aproximar o domínio dos problemas solucionáveis [6, 19, 57, 95].

Entretanto, Rardin e Uzsoy [95] citam quatro métodos para obter um conjunto de instâncias de teste. A seguir, serão detalhados cada um: conjunto de instâncias de teste reais; variações nos conjuntos de instâncias de teste reais; bibliotecas públicas de referência e instâncias geradas aleatoriamente.

4.2.1 Conjunto de Instâncias de Teste Reais

Geralmente, no desenvolvimento de experimentos, os melhores conjuntos de instâncias de testes são provavelmente aqueles que possuem valores reais. São importantes para verificar a efetividade de uma dada heurística, e com isso, obter inferências que serão realizadas em aplicações reais.

Entretanto, pode ser difícil obter muitos casos de testes reais para um experimento computacional. Muitas vezes, empresas ou instituições podem não tornar os dados públicos, ou só aceitam, caso informações como nome e outros dados importantes sejam omitidos. Coletar dados reais também pode levar um tempo considerável.

Os conjuntos de dados reais também têm algumas limitações tais como em relação a conceitos que não foram implementados, ou seja, novos conceitos sobre determinado problema podem requerer informação que não existe em aplicações atuais. Similarmente, conjuntos de dados reais raramente abrangem todas as características do problema. Se os experimentos não testam os limites de variabilidade da heurística, é necessário testar instâncias de vários tamanhos e tipos.

4.2.2 Variações nos Conjuntos de Instâncias de Teste Reais

Uma alternativa, pouco usada, mas que pode aumentar o poder de algumas instâncias reais, isto é, tornar uma instância mais difícil de ser resolvida, é modificar as instâncias que possuem valores reais, com valores aleatórios. A macro-estrutura da aplicação atual deve ser preservada, mas detalhes são modificados aleatoriamente para produzir novas instâncias.

Esta abordagem fixa características estruturais de uma instância conhecida e varia algumas constantes numéricas. Por exemplo, dada uma instância de um problema de roteamento de veículos, podem ser obtidas muitas novas instâncias mudando-se as demandas e deixando-se inalteradas as posições dos depósitos e dos clientes.

4.2.3 Bibliotecas Públicas de Referência

Pesquisas feitas no início do ciclo de vida de um problema muitas vezes apresentam poucas instâncias. Como a pesquisa continua, os conjuntos de instâncias de teste utilizados pelos pioneiros das pesquisas tendem a se tornar coleções de referências clássicas, utilizadas por todos os pesquisadores que trabalham sobre o mesmo problema. Em geral, vários pesquisadores contribuem para a formação de um conjunto de instâncias de referência.

Existem muitos repositórios que possuem instâncias de teste, como QAPLib [11], OR-Library [7, 82], NetLib [79], o TSPLib [108], que contêm instâncias para muitas

classes de problemas clássicos como o Problema de Atribuição Quadrática visto neste trabalho (Seção 6.1), Problema do Caixeiro Viajante, Problema da Mochila, Cobertura de Conjuntos, problemas de *Job Shop Scheduling*, dentre outros.

Todavia, estes conjuntos de referências podem apresentar algumas falhas. A primeira seria que, apesar das primeiras instâncias de referência terem sido construídas com base em informações reais, novas instâncias acrescidas aos conjuntos de referência podem não estar claras quanto a isso e não ser possível saber se foram obtidas de dados reais.

Outra questão é que algumas das instâncias encontradas em bibliotecas de referência não têm o objetivo de representar aplicações reais. Podem servir para avaliar um algoritmo em todos os seus passos, ou simplesmente mostrar um comportamento estranho. Também pode ocorrer que as instâncias escolhidas sejam as que resultaram em bons resultados para os algoritmos desenvolvidos. Isto pode resultar em instâncias com padrões ocultos, ou seja, algum pesquisador construiu instâncias em que foram obtidas boas soluções para seus algoritmos, mas não se sabe se estas instâncias terão bons resultados com outros algoritmos.

4.2.4 Instâncias Geradas Aleatoriamente

Nesta abordagem, as instâncias são geradas de forma totalmente artificial, apesar de suas propriedades poderem ser controladas por parâmetros gerais. É a maneira mais fácil e rápida de se obter um conjunto de instâncias de teste. Entretanto, estas instâncias podem dar conclusões totalmente distorcidas em relação ao mundo real.

As vantagens em gerar instâncias aleatórias são:

- As características do problema estão sobre o controle do pesquisador, se o gerador está devidamente projetado para produzir instâncias com características específicas. Por isso, um bom gerador aleatório pode produzir diversas populações de instâncias, englobando características que muitas vezes não são encontradas em instâncias reais;
- Se um gerador é bem documentado, as características das instâncias geradas serão conhecidas por futuros pesquisadores. Este é o contraste em relação às instâncias de referência, pois as origens destas são obscuras;
- Uma vez que o gerador é construído, este pode fornecer uma quantidade ilimitada de instâncias. Cada semente gerada produz um novo conjunto de dados, mas com as mesmas características implementadas. Muitas vezes é necessário fazer várias replicações para compreender o comportamento de um algoritmo;

- Instâncias construídas por geradores aleatórios são altamente portáteis. Somente o código e a configuração dos parâmetros são necessários para recriar uma grande quantidade de dados;
- Para alguns problemas clássicos, alguns geradores têm sido desenvolvidos, e produzem instâncias em que uma solução ótima é conhecida. Com uma solução ótima em mãos, pode-se avaliar precisamente o quão perto as heurísticas podem alcançar a otimalidade em instâncias de qualquer tamanho.

Entretanto, problemas artificiais são muitas vezes criticados por serem irreais e mais difíceis de serem resolvidos que os problemas reais. Se as instâncias são geradas pelo pesquisador, então o processo de geração deve ser claramente descrito, para ser utilizado por outros pesquisadores [6, 57].

Ao se gerar instâncias aleatórias, deve-se tomar cuidado para não cair em algumas armadilhas. Deve-se saber se os dados gerados são devidamente difíceis (ou seja, abranjam características do problema tais que tornam difícil a busca de soluções) ou representativos, ou se há alguma estrutura em que alguns algoritmos encontram soluções facilmente. Dado um procedimento que gera conjuntos de instâncias, como é possível saber se ele funcionará bem em outro ambiente?

Outra questão a considerar é: quais valores para os parâmetros devem ser testados. Pois geralmente, valores correspondentes de instâncias reais geralmente são desconhecidos ou não documentados. A única escolha é estender o teste-piloto do projeto experimental.

Outro erro que pode ser cometido utilizando-se instâncias aleatórias é concentrar em instâncias não estruturadas, porque além de não refletirem o mundo real, elas podem enganar sobre a dificuldade do problema. Por exemplo, muitos algoritmos para o TSP assimétrico concentram-se em matrizes assimétricas de distâncias com valores que variam de 1 a n , onde n é o número de cidades. Então os pesquisadores geram grandes instâncias para milhares de cidades, mas tem uma grande dificuldade com instâncias estruturadas do TSPLib que contém 53 cidades ou menos [57].

4.2.5 Como Gerar um Conjunto de Instâncias de Teste

Caso forem geradas novas instâncias, este processo de geração deve ser feito sistematicamente, pois não fazer isso pode levar a confusão nos resultados experimentais, podendo até mesmo invalidar todos os resultados do estudo.

Para gerar uma instância, é importante saber quais fatores afetam o desempenho do algoritmo, por isso deve-se gerar instâncias que possibilitem a análise e controle das características da técnica utilizada para resolver o problema. Então, como saber quais fatores são importantes? Esta não é uma questão fácil de responder. Utilizar experiência

e criatividade pode ajudar a inferir conclusões para explicar algum comportamento diferente na execução do algoritmo.

Rardin e Uzsoy [95], ao gerarem suas instâncias para o *job shop scheduling problem*, apresentaram uma lista de parâmetros controláveis e seus respectivos níveis, os intervalos de geração dos números aleatórios e opções de configuração dos parâmetros.

Um método para gerar instâncias aleatórias é basear-se em uma solução ótima conhecida. Com uma solução ótima em mãos, é possível avaliar precisamente o quanto uma solução heurística está próxima de uma solução ótima. Trabalhos utilizando esta abordagem podem ser encontrados em Arthur e Friendewey, 1988 apud Rardin [95], Pilcher e Rardin, 1992 apud Rardin [95], Moscato e Norman, 1998 apud Rardin [95].

Para compreender esta abordagem, suponha um problema modelado em programação linear inteira

$$\left\{ \begin{array}{l} \min cx \\ \text{s. a} \\ Ax = b, \\ x \geq 0, \\ x \in \mathbb{Z}; \end{array} \right.$$

e várias inequações válidas da forma

$$Gx \geq h,$$

que são construídas para manter soluções inteiras viáveis. Então, é construída uma instância seguindo os passos:

1. Gerar coeficientes A e b aleatoriamente, juntamente com uma solução viável x ;
2. Selecionar aleatoriamente um conjunto de inequações válidas $\bar{G}x \geq \bar{h}$, tomando cuidado para incluir somente as inequações que satisfazem a x^* , sendo x^* o melhor valor de solução encontrada;
3. Gerar aleatoriamente multiplicadores $u \geq 0$ duais não-negativos para as principais restrições, sendo $v \geq 0$ para o conjunto de inequações válidas, e $w \geq 0$ para as variáveis primais, com $w_j = 0$ para todo j com $x_j^* > 0$;
4. Calcular o custo do vetor $c = uA + v\bar{G} + w$;
5. Descartar as inequações duais e válidas antes de retornar (A, b, c) , com solução viável x .

Uma solução inteira assim construída é ótima, porque ela é ótima na relaxação linear com as principais restrições e inequações válidas [94].

Os limites para aplicação desta abordagem são a dependência das informações do problema. Para utilizar a abordagem sobre poliedros descrita nos cinco passos anteriores,

é necessário ter um grande e rico conjunto de inequações válidas para o problema e um procedimento que relaciona os conjuntos de inequações a uma determinada solução. É possível que possam ser resolvidos vários problemas combinatórios clássicos, mas esta técnica não se estende a outros modelos aplicados.

4.3 Critérios de Parada

Johnson [57] cita que vários algoritmos de aproximação como a *Multiple-start Local Search* e *Truncated Branch and Bound*, podem dar soluções melhores quanto mais tempo forem executados. Uma ideia poderia ser executar o código durante uma hora e pegar a melhor solução encontrada. Mas fazer isso não renderá resultados reproduzíveis, pois se for utilizado um computador ou sistema operacional diferentes ou algumas diferenças na implementação, pode-se obter soluções com níveis de qualidade diferentes, portanto fazer isto não é recomendável.

Utilizar o tempo como critério de parada e relatar que o tempo de execução foi 60 minutos, por exemplo, pode ser insignificante na tentativa de fazer uma comparação justa, porque estas comparações não são reproduzíveis, pois se os testes forem executados em um computador dez vezes mais rápido, terá uma solução melhor e algumas medidas relativas podem mudar drasticamente.

Para fazer comparações justas, pode-se utilizar como critério de parada alguma medida combinatorial, como o número de vizinhos encontrados e quantidade de passos de *branching*. Desta maneira o algoritmo fica bem definido e o tempo de execução e a qualidade da solução podem ser medidas com um cálculo combinatorial, o qual poderá ser reproduzido futuramente. Com isto, pode-se colocar estes dados em uma tabela de resultados, incluindo também o tempo total de execução de cada algoritmo, e observar como as diferenças provocadas pelas implementações afetam os tempos de execuções (que pode ou não permanecer aproximadamente iguais).

Johnson [57] também sugere que não se use o valor de uma solução ótima como critério de parada. Este erro é encontrado em alguns trabalhos com metaheurísticas em que os algoritmos não têm nenhum critério ou método para verificar a otimalidade, e procuram uma boa solução até algum critério de parada ser atingido. Em alguns algoritmos, o critério de parada é um valor ótimo já conhecido, mas disto surge a questão de que se já é conhecida uma solução, para que perder tempo procurando alguma solução, se a meta de encontrar uma solução ótima já foi alcançada. Se esta abordagem for utilizada, só poderão ser testadas instâncias com valores ótimos já conhecidos, e para as instâncias em que estes valores são desconhecidos este critério não poderia ser utilizado. E para as instâncias em que os valores são conhecidos esse método não refletirá o desempenho

do algoritmo e nem será reproduzível, no sentido de que diferentes tempos de execução podem ser reportados como semelhantes.

Apesar destes problemas, sabe-se que muitas heurísticas são executadas com uma quantidade fixa de passos com o objetivo de obter o melhor resultado, e depois é retornada a melhor solução encontrada. Muitas vezes, quando são estudados algoritmos aleatórios, é fixada a quantidade de execuções que o algoritmo irá realizar, fazendo com que o tempo de execução retornado seja somente o tempo de chegar até a primeira melhor solução. Por este motivo, uma abordagem é apresentar a quantidade de passos ou iterações realizadas antes da melhor solução ser encontrada e, é claro, os tempos de execução também. Com estas informações é possível delinear as regras de como definir os parâmetros que controlam a quantidade total de passos ou iterações executadas [57].

4.4 Execução do Experimento

Segundo Barr et al. [6], deve-se assegurar que o projeto experimental está sendo seguido, pois é nesta fase em que os dados são coletados. Também consideram-se dois fatores ao executar o experimento: a aleatoriedade, que corresponde a realizar os testes em uma ordem aleatória; e o uso do mesmo ambiente computacional.

Johnson [57] afirma que um erro que não se deve cometer é realizar testes com instâncias somente uma vez. Isto pode levar a conclusões erradas e até mesmo tornar o experimento irreprodutível. A quantidade de testes necessários varia de acordo com os objetivos delineados inicialmente. As metaheurísticas podem dar resultados diferentes ao fazer vários testes com uma única instância, principalmente se ela possui componentes aleatórios, como GRASP e Algoritmos Genéticos. Por isto, pode não ser seguro tentar inferir alguma conclusão com um único teste feito sobre uma dada instância.

Para reduzir a variância dos resultados, deve-se testar várias instâncias, e se são utilizados geradores aleatórios, o código deve ser testado sobre cada instância com sementes de inicialização diferentes.

Outro problema citado por Johnson [57] é a utilização do melhor resultado encontrado como critério de avaliação, principalmente quando são estudados algoritmos aleatórios. Geralmente são apresentadas tabelas que contêm o melhor resultado encontrado ou a média dos resultados encontrados. Mas existem dois problemas em relação a estes dados. O primeiro é que a melhor solução encontrada é somente uma amostra da distribuição de soluções existentes, e com isso menos provável de ser reproduzível do que a média. E segundo, os tempos de execução relatados são geralmente uma única execução do algoritmo, ao invés de todas as execuções que foram feitas, e representam apenas o tempo para encontrar a melhor solução. Desta forma, o tempo relatado é um tempo obscuro. Mas se a quantidade de execuções é descrita, o simples procedimento de multiplicar

o tempo de execução pela quantidade de execuções pode superestimar ou subestimar o tempo necessário, sendo que alguns passos necessitam ser feitos apenas uma vez, como por exemplo a leitura da instância e a criação das estruturas de dados, quando são feitos vários testes com uma mesma instância. Uma prática experimental que pode ser feita é realizar o teste k vezes e selecionar a melhor solução, relatar os tempos de execução e as soluções encontradas.

4.5 Ajustes de Parâmetros

De acordo com Johnson [57], existem muitas heurísticas em que os parâmetros precisam ser configurados, como é o exemplo do algoritmo de busca local com múltiplas inicializações (*Multiple-start*)⁶, em que se deve especificar a quantidade de inicializações. E outras heurísticas mais elaboradas como *Simulated Annealing*⁷, Busca Tabu⁸ ou Algoritmos Genéticos⁹ podem necessitar de alguns ajustes nos parâmetros. Este ajuste pode ser feito de duas maneiras: a primeira é fixar os parâmetros para todas as instâncias e a segunda é configurar os parâmetros para cada instância testada. Quando os parâmetros são fixos não há problema em reproduzir o experimento, mas o trabalho deve descrever as configurações utilizadas. Mas se as configurações variam com a instância, deve-se explicar o porquê da utilização destas.

Se determinadas configurações não estão bem especificadas, pode significar que: o algoritmo está mal especificado ou mal implementado, ainda mais se os parâmetros não estão bem explicados, ou se a configuração dos parâmetros demorar um certo tempo não deve ser incluída no tempo de execução, pois se for incluída, pode parecer que o algoritmo levou muito tempo para executar.

Uma regra utilizada é que se valores diferentes para parâmetros são utilizados para diferentes instâncias, o processo de mudança deve ser bem definido, e os ajustes feitos devem ser descritos, bem como o tempo para o ajuste, que deve ser incluído no relatório dos tempos de execução.

Em relação à seleção de parâmetros, pode-se especificar as configurações dos parâmetros e como eles são escolhidos. Os parâmetros associados com alguma regra de parada devem ser documentados e justificados. Outros quesitos incluem: os valores dos

⁶O algoritmo de busca *Multiple-start* consiste em duas fases, a de construção e a de melhoramento das soluções. Ambas fases são realizadas sucessivas vezes com o objetivo de encontrar a melhor solução possível. Portanto, um parâmetro que deve ser passado para o método é justamente a quantidade de vezes que as fases serão repetidas.

⁷Para mais detalhes da técnica *Simulated Annealing*, ver 2.3.2.

⁸Para mais detalhes da técnica *Busca Tabu*, ver 2.3.3.

⁹Para mais detalhes sobre Algoritmos Genéticos, ver 2.3.5.

parâmetros, ou regras, usados para resolver cada instância do problema, onde os valores dos parâmetros diferem para instâncias diferentes [6, 95].

4.6 Questões de Implementação

Para Johnson [57], fazer uma implementação eficiente parece ser óbvio, visto que a eficiência é um dos principais objetivos na área de projeto de algoritmos. Sabe-se que uma implementação eficiente pode dar bons resultados quando são utilizadas estruturas de dados sofisticadas e truques para acelerar o processamento. Isto pode se tornar uma barreira quando se quer mostrar novas ideias de algoritmos, porque para isso devem ser implementadas todas as técnicas possíveis para otimizar o código.

Johnson [57] ainda sugere que um erro que não deve-se cometer é afirmar que o tempo para programar foi curto, que faltou habilidade para programar ou comparar uma implementação otimizada com outra que não é. Por exemplo, afirmar que a implementação poderia ser competitiva com os algoritmos publicados se tivesse mais tempo para implementação ou habilidade para usar as técnicas de otimização de código. Em relação à implementação otimizada, os argumentos podem ser suspeitos, primeiro porque é difícil quantificar o ganho na velocidade obtido através de mecanismos adicionais de otimização de código, a menos que estas diferenças sejam implementadas e testadas. E segundo, não há como garantir que um mecanismo de velocidade será tão significativo se utilizado em outras abordagens de algoritmos.

Em geral, algumas vantagens da eficiência são as possíveis afirmações de competitividade, pois implementações rápidas em geral permitem realizar experimentos sobre grandes instâncias e até mesmo terminar o estudo mais rápido. Mas este princípio não precisa ser seguido minuciosamente, e também não há necessidade de implementar todos os refinamentos teóricos que produzirão pouca melhoria no pior caso, ou seja, que não fazem muita diferença na prática. A menos que um dos objetivos seja avaliar o impacto do tempo de execução com refinamentos teóricos e truques para diminuir a velocidade [57].

4.7 Tempo Gasto na Execução do Experimento

Quando um experimento é executado, deve-se dar atenção ao tempo despendido para realização do experimento, pois deve-se evitar dedicar muito tempo de computação a questões erradas. O tempo computacional é visto somente como o tempo da execução do programa que é dado pelo sistema operacional, entretanto, ele envolve o tempo gasto na criação, execução, e avaliação dos experimentos. Deve-se atentar a alguns desperdícios de tempo que podem ser evitados, como: ao invés de estudar excessivamente uma ou

duas instâncias é melhor utilizar o tempo para testar várias instâncias de uma maneira sistemática [57].

Pode-se contornar este problema realizando experimentos, enquanto o código é executado e depurado. Algumas avaliações que podem ser feitas são: tentar confirmar se o algoritmo supera os outros que foram publicados, examinar como o desempenho varia com o tipo e tamanho da instância; verificar quais são os gargalos computacionais do algoritmo, quais tipos de *tradeoffs* estão envolvidos ou se a escolha das estruturas de dados foram adequadas. As anormalidades no desempenho podem ser interessantes porque se reproduzíveis podem ser alvo de estudos futuros.

Uma alternativa utilizada por Jonhson [57] é utilizar uma abordagem iterativa, que consiste em três passos:

1. Utilizar a primeira metade do tempo proposto de experimentação para gerar grandes quantidades de dados e procurar padrões e anormalidades;
2. Baseado no primeiro passo, finalizar a implementação do algoritmo e decidir quais são as questões interessantes;
3. Analisar os dados resultantes. Se ocorrer alguma falha em responder alguma questão ou surgir outras questões, voltar ao passo 2.

Johnson [57] também sugere que para não desperdiçar tempo em questões erradas com aspectos irrelevantes deve-se utilizar algumas técnicas para ajudar a diminuir o tempo utilizado para execução do experimento, mas que também se obtenha respostas amplas e precisas. A primeira delas é utilizar técnicas de redução de variância ¹⁰. Quando estão sendo estudados algoritmos aleatórios ou instâncias aleatórias, a variância na execução do algoritmo pode produzir resultados equivocados, tornando difícil tirar conclusões.

A segunda é que em um experimento pode-se ter que realizar testes com várias instâncias para obter estimativas confiáveis do desempenho médio do algoritmo. Para obter esse desempenho, suponha que o estudo a ser feito compara o desempenho médio para diversos algoritmos aproximativos e são utilizadas instâncias geradas aleatoriamente. Ao invés de fazer a estimativa do desempenho médio de cada algoritmo sobre uma classe de instâncias de forma independente e então comparar as estimativas, deve-se utilizar o mesmo conjunto de instâncias geradas aleatoriamente para todos os testes. Uma vez que cada algoritmo trabalha com as mesmas instâncias, a variância entre instâncias não precisará ser comparada. Assim, mesmo que não se obtenha intervalos de confiança suficientes, sobre a qualidade das médias das soluções dos algoritmos estudados, para diferenciá-los estatisticamente, a estimativa da diferença média entre os valores de solução pode ser significativa e permitir afirmar que um é melhor que o outro.

¹⁰Para técnicas de redução de variância ler [66].

Johnson [57] ainda cita que fazer a documentação dos programas pode reduzir o tempo gasto na realização do experimento. E quando são utilizadas grande quantidade de dados, deve-se tratar este problema de organizar os dados, ainda mais se eles serão utilizados no futuro. Salvar os dados em arquivos e diretórios com nomes descritivos pode ajudar, e armazenar arquivos com informações relevantes, tais como caminhos, comandos e outras informações. Dados que não podem ser interpretados futuramente são dados inúteis, por isso, deve-se fazer arquivos de saída que contêm todas as informações do experimento, como o tempo de execução, qualidade da solução, nome e versão do algoritmo utilizado, o computador que foi usado (e data, caso o computador tenha recebido alguma atualização), o nome da instância que foi utilizada, as configurações de todos os parâmetros variáveis, e as medidas auxiliares que podem ser relevantes McGeoch, 2001 apud [57].

Análise de Dados e Relato do Experimento

Após a realização dos passos de definição dos objetivos, da escolha das medidas de desempenho, fatores a explorar (vistos no Capítulo 3) e do projeto e execução do experimento computacional (vistos no Capítulo 4), o resultado destas fases, geralmente, é uma grande quantidade de dados para analisar. As medidas de desempenho definidas no início de um experimento podem resultar em um conjunto de dados, obtidos pela execução de diversos algoritmos sobre um conjunto de instâncias de teste.

Rardin e Uzsoy [95] definem dois estágios a serem realizados depois da execução dos testes. No primeiro, Análise dos Dados, descrita na Seção 5.1, busca-se compreender os resultados obtidos, investigar fenômenos anômalos nos dados e verificar se os dados estão corretos. No segundo, Relato dos Resultados do Experimento descrito da Seção 5.2, procura-se esclarecer e mostrar os principais resultados de uma maneira concisa e adequada para publicação. Para melhor compreensão deste Capítulo é necessário ler inicialmente o Apêndice A.

Por fim, a Seção 5.3 apresenta o *checklist* desenvolvido, em que foram listados e organizados todos os itens recomendados neste trabalho para a condução e relato de experimentos.

5.1 Análise dos Dados

Nesta fase os dados coletados são analisados e interpretados, fazendo com que sejam transformados em informações. A análise de dados consiste em avaliar os dados que foram obtidos, aplicando técnicas estatísticas e não estatísticas com relação aos objetivos definidos no início do experimento [6].

Os passos de análise e interpretação são culminantes para as atividades de planejamento e implementação, que ao final determinam todo o mérito do trabalho. Ferramentas de análise de dados, como bibliotecas estatísticas e/ou programas de visualização de dados, são muito úteis. Alguns programas estatísticos muito utilizados são o Matlab, com o pacote *Statistics Toolbox* [65], Minitab [73], Octave [80], Scilab [100], sendo todos

multiplataforma, exceto o Minitab, que é para *Windows*. O único que é *software* livre é o Octave, o restante são programas pagos.

Uma vez que os dados foram analisados, os resultados são interpretados com uma série de conclusões e inferências, deduzidas das evidências coletadas. As recomendações frequentemente incluem futuros experimentos. Portanto, as análises devem ser feitas observando alguns critérios. Algumas das principais medidas de análise são o valor da solução encontrada, médias encontradas pela execução de vários testes, diferenças entre valor encontrado e solução ótima, valores máximo e mínimo encontrados, desvio padrão, erro amostral e tamanho da amostra utilizada [19]. Pode-se considerar os *tradeoffs* chaves, como a qualidade da solução em relação ao tempo e velocidade em relação à robustez. Por exemplo, a taxa de crescimento do tempo para encontrar uma solução em relação ao tamanho do problema pode ser bastante utilizada para mostrar como uma heurística resolve uma certa quantidade de instâncias.

Outro fator a ser observado é o tipo das instâncias em estudo. Por exemplo, para um conjunto fixo de instâncias de teste, pode-se calcular a média e o desvio padrão da qualidade das soluções encontradas com cada algoritmo e então fazer a comparação do desempenho dos algoritmos. Como a população de instâncias é fixa, não há erros nestas estatísticas. Pode haver algum erro nas medidas de tempo de execução, que podem variar de acordo com o ambiente computacional, mas os valores das soluções encontradas permaneceriam os mesmos, caso fossem testados novamente sobre o mesmo conjunto de instâncias de teste. Assim, como não há incerteza na amostragem, não são necessários métodos estatísticos para análise destes resultados [95].

Já a incerteza existe nos experimentos computacionais quando os resultados podem ser vistos como uma amostra aleatória, grande e com uma população essencialmente infinita. Já que não há como testar toda a população, é possível beneficiar-se dos métodos estatísticos, para se ter uma ideia de quão confiáveis são os dados.

A aleatoriedade está presente nos experimentos computacionais quando instâncias são construídas com geradores aleatórios, ou quando algoritmos tomam decisões aleatórias em sua busca. Metaheurísticas como Algoritmos Genéticos e GRASP, por exemplo, tomam decisões aleatórias em cada etapa, e em cada decisão é utilizada apenas uma amostra da população de resultados para cada combinação de instância e algoritmo.

Quando resultados computacionais são obtidos de amostras aleatórias de uma população grande, qualquer análise deve atentar para a possibilidade de que efeitos aparentes nos resultados não sejam simplesmente acidentes estatísticos. Para descobrir se o efeito ocorrido é significativo ou não, é utilizada a técnica de **significância estatística**, que representa a probabilidade de erro envolvida em aceitar um resultado observado como válido, ou seja, representa um índice decrescente da confiabilidade de um resultado, portanto, quanto mais alto o nível de significância, menos se pode acreditar que a relação

observada entre as variáveis da amostra representa um indicador confiável da relação entre as respectivas variáveis na população. Alguns trabalhos com uso de métodos estatísticos, tais como os de Barr et al. [6], Hooker [54] e McGeoch [67], defendem o uso de testes formais de significância estatística como uma maneira de introduzir mais precisão científica nas investigações empíricas de algoritmos.

De fato, é importante que se faça uma análise dos dados, pois podem conter erros amostrais. Mesmo assim, é fundamental ter em mente as limitações dos testes formais de significância estatística. Primeiro, todo teste é construído sob uma série de suposições que podem muitas vezes serem questionadas quando se trata de experimentos com heurísticas. Certamente, estatísticas formais não podem render conclusões muito úteis, a menos que sejam feitas com cuidado e corretamente. Um problema que surge é a dificuldade em trabalhar com os métodos estatísticos e dados computacionais, pois é necessário experiência para organizar os dados e selecionar os métodos corretos a serem utilizados para análise.

O conceito de significância estatística é geralmente confundido com **significância prática**, que diz respeito a se um efeito observado tem realmente importância. O menor dos efeitos pode ter significância estatística se muitos pontos são utilizados para diminuir a probabilidade de erros aleatórios. Isto não significa que o efeito seja importante, ou tenha mérito de ser colocado nos resultados. Um efeito significativo na prática não deve ser afirmado a não ser que tenha significância estatística, porque pode ser um acidente aleatório. Contudo, a demonstração de significância estatística não prova a importância prática. A análise deve ser feita com muito cuidado para não deixar que os testes formais dominem as investigações ou substituam as conclusões e inferências sobre o problema e algoritmos em estudo.

A noção de uma amostra pode ser estendida para abranger conjuntos de dados extraídos de aplicações reais. Se as instâncias forem coletadas de uma forma suficientemente aleatória, pode-se assumir que estas constituem uma amostra aleatória de todas as entradas possíveis. Ainda assim, os métodos estatísticos que consideram a aleatoriedade devem ser aplicados com cautela em tais casos, porque qualquer detalhe na amostragem pode refletir nos resultados.

Para descrever o comportamento de uma amostra, calculam-se as estatísticas descritivas, o valor dos **estimadores** e estimam-se suas variabilidades, e também calculam-se os **intervalos de confiança** [81]. A estatística descritiva consiste em descrever o valor mínimo, valor máximo, média, variabilidade, número de elementos, dentre outros. Este primeiro passo dá uma visão inicial da amostra em questão.

Os estimadores mais utilizados são o **estimador da proporção populacional**, o **estimador da média populacional** e o **estimador do total populacional**. O estimador da proporção populacional é representado por $P_{amostral} = \hat{p}$. O estimador da média popu-

lacional encontrada na amostra $\bar{x}$ é representado por μ . O estimador do total populacional é a multiplicação da média encontrada na amostra pelo tamanho N da população, ou seja, $N\bar{x} = \hat{T}$.

Para obter um intervalo de confiança, deve-se calcular o estimador da variância dos estimadores, pois o cálculo do intervalo de confiança depende da variabilidade do estimador. Dessa forma, dados o tamanho da amostra n , o tamanho da população N , a média populacional $\bar{x}$ e a variância amostral s^2 , calcula-se:

1. Dimensão populacional desconhecida (ou infinita) ou amostragem com reposição: $Var(\bar{x}) = \frac{s^2}{n}$;
2. Dimensão populacional conhecida e finita em amostragem sem reposição: $Var(\bar{x}) = \frac{s^2}{n} \left(\frac{N-n}{N-1} \right)$;
3. Estimador da variância do estimador total: $Var(\hat{T}) = Var(N\bar{x}) = N^2 Var(\bar{x})$.

A avaliação da eficiência de um estimador mostra quão eficiente foi o resultado do trabalho amostral. Assim, dados dois estimadores θ_1 e θ_2 de um mesmo parâmetro, será mais **eficiente** aquele cuja variância for menor, ou seja, se $Var(\theta_1) < Var(\theta_2)$, então θ_1 é mais eficiente que θ_2 .

A análise pode ser feita de várias maneiras, mas em geral a melhor é a que leva a um estimador com menor variância. Com isso, é possível calcular os intervalos de confiança. Primeiro, deve-se saber qual é o tamanho da amostra. Para amostras pequenas (menor que 30 elementos), deve ser empregado o fator t , chamado de distribuição de *t-student*. Para valores grandes, usa-se o fator z [81].

O fator t depende não somente do nível de confiança, mas também do **número de graus de liberdade**, dado por $n - 1$ (n é o tamanho da amostra). Para mais detalhes sobre o fator t , ver [8, 21, 107].

A seguir, são dados os procedimentos para construção de um intervalo de confiança:

1. Definir o nível de confiança que se deseja construir um intervalo de confiança. Isto é importante pois o cálculo do fator (z ou t) depende do nível de confiança utilizado. Para isso, é empregado o mesmo nível de confiança que já havia sido estipulado para o cálculo do tamanho da amostra.
2. Estimar o parâmetro θ .
3. Calcular a estimativa da variância do estimador do parâmetro, $Var(\theta)$.
4. Calcular a raiz quadrada do valor encontrado no passo 3, que é a estimativa do desvio padrão da estimativa do parâmetro, chamado de erro padrão, $(\sqrt{Var(\theta)})$.
5. Calcular o erro amostral resultante da multiplicação do erro padrão pelo fator z ou t , dependendo do tamanho da amostra: $\epsilon = \text{Erro Amostr} = \text{fator} \cdot \sqrt{Var(\theta)}$.

6. Calcular o limite inferior de um intervalo de confiança: $LI_{IC} = \theta - \epsilon$. E calcular o limite superior de um intervalo de confiança: $LS_{IC} = \theta + \epsilon$.

Finalmente tem-se o intervalo de confiança dado por: $IC = \theta \pm \epsilon$).

Já a **análise de variância de um fator** (ANOVA), discutida na Seção 4.1 e detalhada no Apêndice A.4.1, é uma técnica estatística que pode ajudar na análise dos resultados. A ideia básica de ANOVA é assumir que toda variação não aleatória nas observações experimentais é devida às diferenças de desempenho médio nos níveis alternativos dos fatores experimentais [75].

Depois da aplicação do método estatístico ANOVA, pode-se inferir diversas conclusões a respeito dos dados. Para ajudar na compreensão de resultados de experimentos computacionais, a técnica pode ser aplicada para ajudar na fase de análise da pesquisa. As principais deduções em ANOVA são os principais fatores experimentais que parecem ter maior significância estatística, que podem ser os algoritmos ou o tamanho das instâncias, por exemplo. Um indício de significância estatística é somente um sinal inicial, não uma conclusão da investigação. É preciso ver onde os efeitos estão ocorrendo, e se eles têm tanto significância prática como estatística.

Os fatores que não estão envolvidos em interações significativas devem ser analisados se parecem ser insignificantes. Um baixo fator de significância sugere diferenças entre os vários níveis, se são estatisticamente distinguíveis. Fatores que caem nesta categoria podem ser unidos para simplificar a apresentação final.

Uma das principais fontes de informação de ANOVA são as interações que mostram o impacto de como um fator depende do nível de outro. Algo interessante é mostrar as diferenças percentuais entre as médias encontradas.

A comparação de médias também é importante. Se um fator possui somente dois níveis, e os fatores são relativamente significantes, pelo cálculo de ANOVA, as medidas dos níveis são diferentes. Uma indicação da significância de um fator não afirma nada concreto sobre ele, pois pode ser devido ao acaso.

Outra análise, encontrada em [95], é a abordagem de estimativa estatística de valores ótimos. A seguir, a Subseção 5.1.1 descreve como é o procedimento para fazer esta análise.

5.1.1 Estimativas Estatística de Valores Ótimos

Como instâncias de problemas de otimização combinatória geralmente possuem uma grande quantidade de soluções viáveis, a ideia das técnicas de estimativa estatística para valores ótimos é usar uma amostra de soluções para predizer qual é o verdadeiro ótimo. Nos trabalhos (Klein, 1975; Dannenbring, 1977; Golden, 1978; Golden e Alt,

1979; Derigs, 1985;) apud [95] e [42, 63], podem ser encontradas aplicações de tais estimativas.

O método de estimativa mais popular é o proposto por Fisher e Tippett [32], uma distribuição que possui os valores: uma distribuição comum de números reais maiores ou iguais a a , com m variáveis aleatórias, usando a distribuição de Weibull com uma função de distribuição cumulativa $f(z) = 1 - \exp(-[\frac{z-a}{b}]^c)$.

O limite inferior da distribuição a é o parâmetro de localização da distribuição Weibull, b é o parâmetro de escala, e c caracteriza a sua forma. A variável m deve ser grande para garantir maior precisão.

Métodos de previsão da solução ótima (para problemas de minimização) exploram esta fórmula como valores de uma função objetivo com soluções factíveis para um modelo de otimização como pontos de uma distribuição de um valor objetivo. Assumindo que é possível igualar as variáveis aleatórias contínuas resultantes da função de Fisher-Tippett, mas utilizando uma distribuição discreta, um conjunto de m variáveis aleatórias deve ser aproximada para a distribuição Weibull. Se n amostras independentes da mesma instância são selecionadas, elas podem ser combinadas para estimar os parâmetros da distribuição assintótica. Em particular, pode-se aproximar o parâmetro a , que é o valor de uma solução ótima da instância, para avaliar o desempenho da heurística.

Existem várias alternativas para estimar os parâmetros de Weibull, mas uma maneira de obter resultados satisfatórios é a seguinte: dado um conjunto de variáveis independentes, classificadas por ordem decrescente, $z_{[1]} \geq z_{[2]} \geq \dots \geq z_{[n]}$, o parâmetro de locação (*location parameter*) ou valor ótimo a pode ser estimado como:

$$\hat{a} = \frac{z_{[1]}z_{[n]} - (z_{[2]})^2}{z_{[1]} + z_{[n]} - 2z_{[2]}}. \quad (5-1)$$

Uma estimativa de confiança para o ótimo deveria ser significativa por si só, entretanto, Golden e Alt, 1979 apud [95] definiram um intervalo de confiança mais seguro z_l que deve ser menor ou igual ao valor ótimo s^* com probabilidade alta. Em outras palavras, o parâmetro b é estimado de $\hat{a}$ como

$$\hat{b} = z_{[0.63n+1]} - \hat{a}, \quad (5-2)$$

e então calcula-se

$$z_l = \hat{a} - \hat{b}. \quad (5-3)$$

Teoricamente, este intervalo deve cobrir o valor ótimo s^* com probabilidade aproximada de $1 - e^{-n}$.

Um modo simples de obter uma amostra mínima z_i é selecionar n soluções iniciais aleatórias do problema de otimização em questão e encontrar cada ótimo local

com uma busca simples. Estes n mínimos locais, que são os melhores da vizinhança, são usados como z_i . As inicializações aleatórias fazem com que cada z_i seja independente.

Existem outras abordagens, como por exemplo, o caso em que cada elemento z_i é uma sequência de soluções encontradas pela metaheurística *Simulated Annealing* [84].

O teste de hipóteses estabelece quais os pontos que são aproximadamente independentes. Outro exemplo é tomar como grupos mínimos z_i partes de soluções encontradas com a metaheurística Algoritmos Genéticos [36].

Estas técnicas de estimativas são consideradas promissoras, pois os esforços computacionais para calcular estimativas de valores ótimos e limites de confiança são relativamente pequenos, e as técnicas oferecem a esperança de obter informação confiável sobre valores de soluções ótimas que são independentes do problema de domínio.

5.2 Relato dos Resultados dos Experimentos

No âmbito de conseguir demonstrar que a pesquisa feita tem alguma contribuição, o leitor deve ser convencido disso através de um relatório feito com qualidade. Crowder et al. [20] apresentam alguns itens importantes que devem ser relatados nos resultados. Uma sugestão é apresentar as conclusões baseadas nos dados e resultados, descrever conclusões bem justificadas é sem dúvida, um dos principais objetivos de experimentos com algoritmos, isto é, aprender algo sobre o comportamento do algoritmo em estudo [57].

As declarações feitas sem a devida justificativa devem ser claramente rotuladas como especulação por parte dos autores. A justificativa para os indicadores de desempenho, como por exemplo, tempo de execução ou robustez, devem estar no contexto das metas do experimento. Se o tempo de execução é usado como medida, então é necessário uma descrição precisa de como estes tempos foram computados e se eles incluem os procedimentos de entrada e saída. Pode-se também apresentar tempo de pré-processamento, se houver.

Outros aspectos importantes são os critérios de parada, incluindo todas as regras, e análise da quantidade de trabalho por iteração, se o número de iterações é usado como medida de desempenho.

Em relação ao conjunto de instâncias de teste, deve-se relatar também as dificuldades dos problemas estudados, como uma instância que não pôde ser resolvida pelo algoritmo desenvolvido, limitações de *software* ou *hardware*, ou outra causa de falha.

Se forem comparados vários métodos, alguns tópicos são úteis, tais como: critério de convergência; tolerâncias; requisitos de armazenamento; pontos de partida; ambiente computacional e métodos para padronizar resultados.

Itens opcionais seriam, por exemplo, citar os efeitos das tolerâncias sobre o tempo de execução, robustez e quantidade de iterações; efeitos sobre critérios de parada

diferentes; e também medidas de variação do desempenho devido a diversas táticas internas, como por exemplo, várias maneiras de gerar uma solução inicial.

Ao se utilizar amostragem, Oliveira [81] cita que é imprescindível informar qual a população utilizada; deve-se deixar claro o que se buscou estimar: médias, totais ou diferenças; citar o tamanho da amostra examinada; quando a amostra for estudada por um enfoque estatístico, informar a margem de erro e o nível de confiança utilizados, e obviamente, descrever o método que foi aplicado para a seleção dos elementos da amostra. Não há necessidade de mostrar o erro amostral, pois seu valor está embutido no intervalo de confiança calculado.

Johnson [57] menciona algumas perguntas importantes que podem ser respondidas a fim de contribuir para a compreensão do experimento:

1. Como os detalhes de implementação, configuração de parâmetros, heurísticas, e as escolhas de estruturas de dados afetam o tempo de execução do algoritmo?
2. Como o tempo de execução é comparado com o tamanho da instância e quanto ele depende da estrutura da instância?
3. Qual operação algorítmica ajuda a explicar o tempo de execução?
4. Quais são os gargalos computacionais na prática, e como eles dependem do tamanho da instância? Como isso difere das análises de pior caso?
5. Como o tempo de execução é afetado pelo computador utilizado?
6. Qual a variância encontrada quando comparadas instâncias similares ou iguais no mesmo computador?
7. Como é o tempo de execução do algoritmo comparado com seus principais concorrentes, como estas comparações são afetadas pelo tamanho da instância e sua estrutura, ou arquitetura da máquina, e como as diferenças podem ser explicadas em termos de contagem de operações?
8. Dada uma nova classe de instâncias que tenha sido identificada, ela causa mudanças significativas no comportamento do algorítmico para os algoritmos já estudados?

Por conseguinte, com o intuito de responder às questões apontadas, a Subseção 5.2.1 fala sobre a apresentação de resultados, a Subseção 5.2.2 descreve como relatar a variância, a Subseção 5.2.3 fala sobre os principais requisitos que um experimento deve oferecer, a reprodução e comparação. Por fim, a Subseção 5.2.4 aponta algumas falhas que podem ser evitadas no relato dos resultados.

5.2.1 Apresentação dos Resultados

Um dos problemas encontrados em artigos e relatórios é a apresentação de dados sem interpretação. Não é suficiente realizar os testes, colocar os resultados em tabelas e

deixar que o leitor tire suas próprias conclusões [57, 68]. No mínimo devem ser relatados os padrões encontrados nos dados. Se as questões forem bem delineadas, o experimento dará alguma resposta. Uma justificativa sem fundamentos seria, por exemplo, afirmar que um algoritmo é melhor que outro porque levou metade do tempo para encontrar uma solução, sendo que este algoritmo só se saiu melhor porque foi testado sobre instâncias pequenas, e o outro teve melhores resultados sobre instâncias grandes.

Gráficos oferecem um meio de visualizar todos os dados e uma compreensão melhor pode vir da análise do conjunto de dados completo e não apenas as estatísticas resumidas. No entanto, apesar de gráficos e tabelas serem uma ótima maneira de mostrar as conclusões, não devem ser usados como único meio de apresentar e explicar os resultados. Tabelas adicionais podem ser inseridas como anexos do texto. Porém, usar somente tabelas também não é uma maneira eficiente de apresentar resultados. Portanto, se houver um gráfico para facilitar o entendimento, é preferível. Da mesma maneira, gráficos sem tabelas não possibilitam a compreensão dos dados, pois as figuras permitem dar uma ideia geral, mas não apresentam detalhes dos resultados. Enfim, um bom artigo deve conter tabelas e gráficos.

Johnson [57] dá uma atenção especial em relação a gráficos, cada resultado deve ser diferenciado, para isso pode-se utilizar pontilhados e tracejados diferentes, de modo que as linhas não sejam indecifráveis quando os pontos estão muito próximos ou coincidem. Eles devem ser claros e organizados. Por exemplo, se o trabalho conta com a análise de 20 algoritmos, fica difícil e ilegível colocar todos os algoritmos em um único gráfico, e fazer um gráfico para cada algoritmo poderia gerar confusão. Uma solução seria ter valores múltiplos, com cada valor dedicado a um subconjunto de algoritmos. E fazer outro gráfico que liga estes subconjuntos, fazendo sobreposição de conjuntos, para mostrar as conclusões gerais.

Triola [107], diz que o principal objetivo na construção de um gráfico, é entender bem o conjunto de dados para que se possa utilizar gráficos que representem e revelem características marcantes nos dados. Quando um grande conjunto de dados é analisado, é útil que sejam organizados e resumidos numa tabela de **distribuição de frequência**, que lista os valores dos dados com os suas respectivas frequências. Utilizar esta tabela auxilia a entender a natureza da distribuição dos dados, que pode ser em forma de sino, uniforme ou assimétrica. O histograma é o tipo de gráfico utilizado para representar como é a distribuição de um conjunto de dados. Alguns gráficos mais elaborados são: polígono de frequência, ogiva, gráfico de pontos, diagrama de ramo e folhas, gráficos de pareto, gráfico de setores, diagrama de dispersão e gráficos temporais. Triola explica em que situações usá-los, no Capítulo 2 do seu livro. Cleveland [14] também é uma boa referência sobre como construir gráficos e visualizar informação.

Johnson [57] também cita dicas em relação a tabelas, em que pode ocorrer um

problema chamado “a coluna que falta”. Por exemplo, se estão sendo avaliados algoritmos que têm bons limites inferiores, não basta apresentar apenas os resultados obtidos pelos algoritmos e deixar a tarefa de calcular a divisão que vai mostrar quão perto as soluções ótimas estão dos limites inferiores. O título da coluna também deve ser bem definido, assim como os títulos das figuras, pois um título ambíguo ou enigmático dado a uma coluna ou figura pode tornar a figura ou tabela inútil. Por exemplo, se a coluna que contém os tempos de execução não explicita se os tempos relatados incluem o tempo de leitura da instância ou o pré-processamento, ou mesmo se refere somente ao tempo de uma única execução, o tempo total de todas as execuções ou até mesmo a média dos tempos, ou pior, se não especifica qual a grandeza utilizada, se são microssegundos, segundos ou horas. Outro exemplo é a coluna “número de iterações”, quando em nenhuma parte do trabalho é explicado claramente o que constitui esta iteração. O destaque também é algo importante, pois nem todos os números exibidos têm igual importância. Logo, os dados mais importantes devem ser destacados, como por exemplo, a melhor solução obtida.

Outro detalhe ao apresentar resultados é a quantidade de dígitos significativos. Uma maneira simples de definir quantos dígitos significativos serão publicados, com o objetivo de dar atenção a diferenças importantes e ao mesmo tempo tratar qualquer erro amostral, é arredondar os valores sistematicamente e justificar a quantidade de dígitos antes de expor os resultados [95].

Outra técnica a considerar é a **significância prática**, que mostra quanto um valor pode ser considerado importante para investigações futuras. Por exemplo, uma solução ótima com valor 27.349682, é o mesmo que 27.3. Outra solução com o valor 27.2, pode ser melhor que a anterior, mas na prática não há diferença.

Existem muitas técnicas e maneiras de exibir resultados, que dependem dos objetivos que se quer alcançar. Para ter um olhar mais crítico sobre esta questão da mostra de resultados, é aconselhável procurar trabalhos de referência, tais como o de Triola [107] ou Cleveland [14], além das dicas colocadas neste trabalho.

5.2.2 Relatando a variância

Como foi visto na Seção 4.1, um planejamento experimental básico consiste em organizar os dados em uma tabela, em que as colunas correspondem aos algoritmos testados. Nas linhas, além da solução e do tempo de cada instância, Rardin e Uzsoy [95] apresentam quatro alternativas para mostrar resultados experimentais, mostradas na Tabela 5.1. A primeira, letra (a), relata o desvio padrão e a média amostral. A segunda, letra (b), apresenta o melhor e o pior valores encontrados para determinada instância. Ambas alternativas omitem muitos detalhes. Os dados podem ser mostrados em porcentagem.

Relatar o desvio padrão e a variância¹ é importante porque com estes dados é possível mensurar o grau de heterogeneidade de uma população ou amostra em relação à variável em estudo, ou seja, quão dispersos estão os valores encontrados na população ou amostra [81]. Por exemplo, a variável solução deve apresentar uma certa variabilidade, caso seja encontrada por métodos heurísticos com componentes aleatórios, como Algoritmos Genéticos ou GRASP.

Na letra (c), o erro padrão serve para mostrar que, como uma amostra possui um conjunto restrito de elementos, precisa-se saber o quão imprecisas são as médias encontradas. Por isso são calculadas margens de erro, para controlar a imprecisão dos resultados, com os respectivos níveis de confiança.

Por fim, na letra (d) são dados os intervalos de confiança, juntamente com a média amostral. Os intervalos de confiança são utilizados para indicar a confiabilidade de uma dada estimativa, isto é, mostram quanto os resultados encontrados são confiáveis, pois em vez de estimar somente um valor para um parâmetro, é calculada uma faixa de valores em que o valor do parâmetro pode estar. A Seção 5.1 mostra como calcular o intervalo de confiança.

	Algoritmo 1	Algoritmo 2	...	Algoritmo n
(a) Desvio padrão Média Amostral Desvio padrão				
(b) Intervalos Média Amostral Pior Valor Encontrado Melhor Valor Encontrado				
(c) Erro Padrão da Média Média Amostral Erro Padrão da Média				
(d) Intervalos de Confiança Média Amostral Intervalo de Confiança				

Tabela 5.1: Medidas de variância em uma tabela. Baseado em [95].

5.2.3 Reprodução e Comparação do Experimento

Segundo Crowder, Dembo e Mulvey [20], as tecnologias computacionais estão sempre evoluindo rapidamente, tornando praticamente impossível reproduzir um experimento computacional, pois pode-se até utilizar o mesmo ambiente computacional, mas ao mudar a versão do compilador ou a versão do sistema operacional pode resultar em diferentes sequências de operações que influenciam no resultado final. Portanto, quando é

¹Ver Seção A.3, no Apêndice A.

necessária a reprodução do experimento, não significa que é uma reprodução precisa dos resultados. Ao contrário, um conjunto de resultados que coincide com o original dentro de um limite pode ser atribuído a mudanças na tecnologia. Mas em muitos casos, é difícil prover informação suficiente em um artigo, de modo a permitir que o leitor possa reproduzir todos os resultados apresentados. Uma solução útil seria descrever uma lista detalhada dos dados, suficiente para reproduzir o experimento computacional. Entretanto, os critérios devem ser completos, sensatos, e justificados cientificamente para que os próprios autores sejam capazes de replicar o experimento.

Dentre as qualidades que um bom artigo ou relatório deve ter, está a questão da **reprodução do experimento**. Segundo Johnson [57], na reprodução de um estudo, um cientista pode usar os mesmos métodos básicos, mas usar diferentes aparatos, materiais distintos mas similares e, possivelmente, diferentes técnicas de medição. Os dados serão reproduzíveis se os resultados dos dados originais obtidos são consistentes com o do experimento e apresentam as mesmas conclusões.

Para um experimento ser passível de reprodução, deve ser documentado cuidadosamente. Isto inclui o relato detalhado do modelo experimental utilizado, o relato dos testes, detalhes sobre os algoritmos desenvolvidos e implementação, descritos em detalhes suficientes que permitam a replicação [6, 19, 57, 95].

O relato do modelo experimental inclui, além de sua descrição e justificativas da utilização deste, os objetivos do experimento, como foram feitos os testes, incluindo dados como a quantidade de testes repetidos por instância, quantidade de instâncias testadas, quais e quantas sementes foram utilizadas, quantidade máxima de iterações, critério de parada, quais foram os critérios para encontrar uma solução inicial, quais foram os critérios de parada do algoritmo utilizados, quais foram os valores dos parâmetros, tanto da heurística quanto do algoritmo.

A descrição do algoritmo é um dos requisitos mais importantes na reprodução de um experimento. Deve-se dar a descrição completa do algoritmo, esclarecer qual classe do problema é estudada e quais instâncias ele encontra soluções. Também podem ser apresentadas uma análise da complexidade do algoritmo, apresentar análise de esforço computacional gasto pelo algoritmo em cada iteração.

Detalhes de implementação são úteis para que outros pesquisadores possam ao menos fazer de forma similar e chegar às mesmas conclusões. Além disso, qualquer código desenvolvido também melhora o mérito científico do trabalho. Detalhes como linguagem de programação, descrição dos dados de entrada, das configurações dos parâmetros, técnicas de pré-processamento, descrição da estratégia inicial, dados sobre compilador e opções, onde é possível encontrar o código-fonte, e como utilizá-lo, são itens importantes em um relato.

Rardin e Uzsoy [95], McGeoch e Moret [68] afirmam que não há razão para

que os relatórios publicados incluam todos os detalhes para reproduzir o estudo. Contudo, estas informações devem ser detalhadas em algum lugar, como um documento de trabalho, memorando ou relatório técnico, que contenha todos os detalhes necessários para recuperar os resultados, que incluem: instâncias testadas, configuração de parâmetros e códigos-fonte de todos os algoritmos testados, inclusive o de geração de instâncias aleatórias.

Como o ambiente de teste, que é o computador, pode influenciar no desempenho do algoritmo, alguns itens devem ser documentados: modelo e marca do computador; quantidade, tipos e velocidades dos processadores; tamanho e configuração das memórias *cache*, *swap* e principal; sistema operacional e versão; linguagens de programação, compiladores e suas configurações, e demais bibliotecas utilizadas [68].

Alguns padrões se utilizados, serão irreproduzíveis. Dentre as questões que dificultam a reprodutibilidade de um experimento, enquadram-se [57, 67]:

- **Relatar somente o valor da solução:** Torna o experimento irreproduzível em um sentido limitado, mas não no sentido mais amplo em que pode-se realizar experiências em casos semelhantes para comparar se os resultados são semelhantes;
- **Relatar somente a porcentagem sobre a melhor solução calculada:** Não representa muito, devido ao fato de que se forem calculadas várias soluções, não há como afirmar que elas são as melhores. Para isso, deve-se sempre fornecer a instância ou o valor encontrado;
- **Relatar a porcentagem sobre uma estimativa da solução ótima esperada:** Para instâncias geradas aleatoriamente, estes dados serão reproduzíveis se as estimativas não forem definidas ou se o método não for especificado. Pode-se ter resultados significativos se a estimativa é de fato consistente e perto do ideal esperado, e se os valores ótimos encontrados têm variância relativamente baixa;
- **Relatar a porcentagem excedente do limite inferior:** É reproduzível se o limite inferior pode ser calculado facilmente ou possível de fazer um cálculo aproximado;
- **Relatar o percentual de melhora de alguma heurística:** É reproduzível se a heurística é completamente especificada, apesar de definir uma heurística ser complicado, pois muitos autores geralmente dão um nome à ela, como “2-Opt” ou *Simulated Annealing*. Sabe-se que todos estes algoritmos podem ter variações tendo diversos tipos de comportamentos, e somente o nome da heurística não dá detalhes suficientes para distingui-las. Portanto, basta usar um algoritmo simples como padrão, que possa ser especificado precisamente em poucas palavras, e preferencialmente que seja determinístico. Se um algoritmo mais complicado é necessário, então uma opção viável é disponibilizar o código da implementação, ou usar algum repositório de códigos existente na *Web*.

É desejável que o autor também dê acesso a código, instâncias e outros dados relevantes para futuros pesquisadores, tornando os resultados passíveis de comparação com novos algoritmos ou instâncias construídas. Claro que muitas recomendações utilizadas para deixar o experimento reproduzível devem ser utilizadas, mas alguns procedimentos a mais são necessários [57]. A seguir são citadas algumas sugestões que tendem a facilitar a **comparação do experimento**.

Greenberg [43] afirma que a qualidade da solução é um dos fatores que deve ser demonstrada no relato da pesquisa. Para Barr et. al [6] o pesquisador deve, dentro do possível, medir a acurácia, isto é, a proximidade de uma solução gerada por uma heurística com a solução ótima. Quando uma solução ótima é conhecida, a solução heurística pode ser comparada como uma medida de efetividade da heurística. Quando soluções ótimas são desconhecidas ou não podem ser obtidas por métodos conhecidos, outra medida de desempenho deve ser mostrada pelo pesquisador, como a comparação do limite inferior (superior), ou comparação com valores publicados na literatura. Quando possível, pode ser dada uma conclusão a respeito de como a qualidade da solução mantém-se nas instâncias do problema, crescendo no tamanho ou complexidade.

Quando um dos objetivos da pesquisa é demonstrar que uma heurística particular supera outra em um ou mais fatores, usa-se a comparação estatística dos resultados. Uma vez que mais informação possa ser vinculada, se os resultados obtidos são contrastados com outros métodos, pode-se listar e identificar pontos de referência para fazer comparações. Resultados publicados podem ser usados, desde que bem conhecidos.

É válido lembrar que comparações como a qualidade da solução versus esforços computacionais e outras comparações, como tempo versus tamanho do problema, e robustez e qualidade mostram o comportamento do algoritmo. Outro fator de interesse é a análise custo-benefício, como comparações de tempo e memória, e a contribuição dos estágios individuais de uma heurística multi-fase. E claro, os autores também devem destacar os resultados inesperados ou estranhos. Sempre que possível, deve tentar explicá-los, caso contrário, devem ser apresentados como problemas dignos de uma investigação mais aprofundada.

O último passo a ser realizado é a escrita das conclusões. Com a análise de dados bem feita, é possível justificar as conclusões a partir dos dados que foram apresentados, bem como dar direções das pesquisas futuras, possíveis melhoras no algoritmo, identificação das instâncias que tiveram uma melhora nos resultados e também as que não foram resolvidas ou não alcançaram melhora, problemas e dificuldades encontradas, como por exemplo, casos anômalos ou resultados que saíram fora do padrão que foram obtidos.

5.2.4 Falhas ao Relatar os Resultados

Em relação aos dados brutos, pode ser complicado relatar todos os dados calculados, mas deve-se tomar cuidado na escolha dos dados a serem publicados, pois a maneira que eles são mostrados podem fazer com que o leitor não consiga extrair conclusões satisfatórias e ainda prejudicar a reprodução e comparação do trabalho. Por exemplo, se somente as médias forem relatadas, deve-se citar as instâncias e a quantidade de execuções que foram usadas para calcular as respectivas médias. E se os valores precisos dessas médias são importantes para as conclusões, pode-se fornecer informações sobre a distribuição dos resultados, que podem ser mostrados com desvio padrão, histogramas ou gráficos. Deve-se deixar claro que as médias devem fornecer informações suficientes para que possam ser calculadas novamente, se necessário.

Johnson [57] menciona ainda que um erro muito comum é a apresentação estatística das médias com muitos dígitos de precisão após a vírgula, e conclusões sobre as diferenças encontradas, que na realidade correspondem ao ruído dos dados. E as medidas de tempo de execução relatadas podem ser enganosas, pelo motivo da imprecisão dos métodos para cálculo de tempo da maioria dos sistemas operacionais, incluindo também ações de usuários que influenciam no tempo de execução do experimento. Certamente não é possível ter uma ótima precisão dos dados, por exemplo, em uma tabela que inclui tempos de execução de várias instâncias. Pode-se usar duas unidades após a vírgula para conseguir dígitos de precisão sobre instâncias menores, e deve-se mantê-los mesmo que os resultados das instâncias maiores precisem de seis casas decimais. Uma alternativa seria substituir os dígitos insignificantes com 0, mas explicar com um comentário no texto este detalhe.

Outra preocupação que surge e que deve ser relatada são os resultados anormais, que diferem do valor esperado ou que são inconsistentes com as conclusões desejadas. Estes resultados não devem ser omitidos quando são encontrados. Pode ocorrer que apareça a anomalia e o pesquisador não consiga explicá-la, mas ela deve ser citada da mesma forma, pois elas podem explicar comportamentos importantes dos algoritmos, implementação ou até mesmo características das instâncias testadas. Não deve-se esquecer que a pior anomalia é aquela que o pesquisador não percebe. Isso pode deixar o leitor em dúvida, porque leva-o a pensar que poderia ser ou erro de digitação ou resultado anormal. A principal conclusão a ser tirada sobre anormalidades é que elas são importantes, devem tentar ser compreendidas e não devem ser esquecidas.

Para concluir, Johnson [57] dá também atenção a algumas questões e lembra:

- Nunca confie em um gerador de números aleatórios;
- Nunca confie que seu código está correto;
- Nunca confie que um autor tenha conhecimento de toda a literatura;

- Nunca confie em sua memória a respeito de onde você armazena os dados (e como foram gerados);
- Nunca confie que seu computador permanecerá sem alterações;
- Nunca confie em *backup* ou *sites* que armazenam dados por tempo indeterminado;
- Nunca confie em um perito na análise experimental.

5.3 Checklist para Avaliação de Relato de Experimento Computacional

Para facilitar a análise, foi desenvolvido um *checklist* de relato de experimento computacional, que foi baseado em Crowder, Dembo e Mulvey [19]. O *checklist* original está na Tabela B.1. Após o estudo desenvolvido, com recomendações descritas nos Capítulos 3, 4 e 5, foram adicionados itens ao *checklist*, que resultou na Tabela 5.2. O objetivo do *checklist* é servir como um guia, pois sumariza os itens e medidas necessários em experimentos com algoritmos que foram vistos neste trabalho, auxiliando tanto na condução do experimento quanto no relato.

A Tabela 5.2 proposta é dividida em seis partes, de A a F, organizadas na forma de condução de um experimento, ou seja, a primeira avalia a revisão da literatura, após modelo experimental, apresentação dos algoritmos, implementação, relato e análise dos resultados, e por fim as conclusões. Para cada item, a segunda coluna mostra a referência do texto, para encontrar maiores detalhes que se encontram no presente trabalho. A última coluna, chamada Peso, é dedicada a atribuir valores de acordo com a análise a ser feita.

A - Revisão da Literatura		
Itens Recomendados	Referência	Peso
1) O problema é novo?	Seção 3.3	
2) O algoritmo proposto é novo?	Seção 3.3	
3) O algoritmo já foi implementado?	Seção 3.3	
4) O algoritmo já foi estudado para o problema em questão?	Seção 3.3	
5) Fala sobre modelagens existentes?	Seção 3.3	
6) Fala sobre métodos já desenvolvidos para o problema?	Seção 3.3	
7) Definição clara do problema.	Seção 3.3	

B - Modelo Experimental			
Itens Recomendados		Referência	Peso
1) Definição clara dos objetivos do experimento	a) Comparar uma abordagem com técnicas já implementadas	Seção 3.4	
	b) Testar e melhorar algoritmos para problemas difíceis	Seção 3.4	
	c) Comparar algoritmos existentes e estruturas de dados para problemas	Seção 3.4	
	d) Comprovar e refinar conjecturas	Seção 3.4	
	e) Desenvolver bibliotecas para algoritmos básicos e estruturas de dados	Seção 3.4	
	f) Desenvolver ferramentas para facilitar o projeto e análise de algoritmos	Seção 3.4	
2) Modelo experimental	a) Básico (instâncias $\times$ algoritmos)	Seção 4.1	
	b) Estatístico	Seção 4.1	
	c) Blocagem de instâncias	Seção 4.1	
	d) Balanceamento de qualidade e tempo	Seção 4.1	
3) Descrição da execução do experimento	a) Quantidade de testes repetidos feitos por instância	Seção 5.2.3	
	b) Quantidade de instâncias testadas	Seção 5.2.3	
	c) Quantidade de sementes utilizadas	Seção 5.2.3	
	d) Quantidade máxima de iterações	Seção 5.2.3	
	e) Critério de parada utilizado	Seção 5.2.3	
4) Descrição do conjunto de instâncias de teste utilizado	a) Instâncias reais ou aleatórias	Seção 4.2	
	b) Se são instâncias de referência	Seção 4.2.3	
5) Geração de novas instâncias	a) Descrição do gerador de instâncias	Seção 4.2.4	

C - Apresentação dos Algoritmos			
Itens Recomendados		Referência	Peso
1) Descrição completa do algoritmo		Seção 5.2.3	
2) Classe do problema que o algoritmo proposto resolve	a) Qual tipo de instâncias o algoritmo encontra soluções?	Seção 5.2.3	
	b) O algoritmo encontra soluções para instâncias de até que tamanho?	Seção 5.2.3	
3) Descrição da técnica de estratégia inicial		Seção 5.2.3	
4) Uso de diferentes critérios de inicialização		Seção 5.2.3	
5) Uso de diferentes critérios de término		Seção 5.2.3	
6) Dados dos parâmetros da heurística (Ex: tamanho da lista tabu)		Seção 5.2.3	
7) Uso de diferentes valores nos parâmetros do algoritmo		Seção 5.2.3	
8) Análise de complexidade do algoritmo		Seção 5.2.3	
9) Análise da quantidade de trabalho por iteração		Seção 5.2.3	

D - Implementação			
Itens Recomendados		Referência	Peso
1) Linguagem de programação		Seção 5.2.3	
2) Descrição dos dados de entrada		Seção 5.2.3	
3) Descrição das configurações		Seção 5.2.3	
4) Descrição de técnicas de pré-processamento		Seção 5.2.3	
5) Armazenamento dos requisitos e estruturas de dados		Seção 5.2.3	
6) Compilador		Seção 5.2.3	
7) Opções do compilador		Seção 5.2.3	
8) Sistema Operacional		Seção 5.2.3	
9) Hardware (modelo do computador, processador e memória)		Seção 5.2.3	
10) Se o código está disponível		Seção 5.2.3	
11) Instruções para uso		Seção 5.2.3	

E - Relato e Análise dos Resultados			
Itens Recomendados		Referência	Peso
1) Medidas de desempenho	a) Qualidade da solução - acurácia com que as soluções são obtidas	Seção 3.5	
	b) Esforço computacional - Tempo da melhor solução encontrada	Seção 3.5	
	c) Esforço computacional - tempo médio total de execução	Seção 3.5	
	d) Esforço computacional - tempo por fase (se existirem fases)	Seção 3.5	
	e) Robustez	Seção 3.5	
	f) Precisão numérica	Seção 3.5	
	g) Quantidade de iterações	Seção 3.5	
	h) Quantidade de chamadas de uma determinada função	Seção 3.5	
	i) Operações matemáticas	Seção 3.5	
2) Justificativa das medidas utilizadas		Seção 5.1	
3) Medidas de análise	a) Valor da solução encontrada	Seção 5.1	
	b) Médias	Seções 5.1, A.3	
	c) Totais	Seção 5.1	
	d) Diferenças	Seção 5.1	
	e) Valor mínimo encontrado	Seção 5.1	
	f) Valor máximo encontrado	Seção 5.1	
	g) Desvio padrão	Seções 5.1, A.3	
	h) Erro amostral	Seções 5.1, A.3	
	i) Tamanho da amostra utilizada	Seções 5.1, A.3	
4) Análise estatística	a) Intervalo de confiança	Seção 5.1	
	b) Nível de confiança	Seções 5.1, A.3	
	c) Margem de erro	Seção A.3	
	d) Tamanho da amostra utilizada	Seção A.3	
5) Uso de gráficos legíveis		Seção 5.2.1	
6) Uso de tabelas legíveis		Seção 5.2.1	

F - Conclusões			
Itens Recomendados		Referência	Peso
1) Justificar as conclusões a partir dos dados apresentados		Seção 5.2	
2) Identificação das instâncias que foram resolvidas com êxito		Seção 5.2	
3) Identificação das instâncias que não foram resolvidas		Seção 5.2	
4) Possíveis melhoras no algoritmo		Seção 5.2	
5) Direções nas pesquisas futuras		Seção 5.2	

Tabela 5.2: *Checklist para Relato de Experimento Computacional Proposto.*

Estudo Exemplo: Problema de Atribuição Quadrática

O objetivo deste capítulo é fazer um estudo sobre o relato dos experimentos computacionais realizados por alguns artigos bastante citados na literatura, mostrando como diferentes autores abordam o relato de seus respectivos experimentos. Estes artigos tratam do Problema de Atribuição Quadrática - PAQ (do inglês *Quadratic Assignment Problem* - QAP), o qual é um problema NP-Difícil, estudado a cerca de aproximadamente 60 anos, e comumente resolvido com métodos heurísticos. As recomendações descritas nos Capítulos 3, 4 e 5 são utilizadas como base na condução do estudo e o *checklist* (Seção 5.3) foi utilizado para guiar e deixar o estudo mais objetivo.

A análise destes trabalhos consistiu na verificação dos itens necessários para compreensão, reprodução e comparação dos experimentos realizados. Os itens foram condensados no *checklist*, considerando como principais pontos a serem analisados: revisão bibliográfica, modelo experimental, apresentação dos algoritmos, implementação, relato dos resultados e conclusões.

Os artigos selecionados estão listados na Tabela 6.1, ordenados pela data de publicação, num intervalo aproximado de seis anos entre os artigos. Vale ressaltar que o primeiro artigo estudado é da década de 80, quando as metaheurísticas começaram a ser difundidas, e uma das primeiras aplicações da metaheurística *Simulated Annealing* ao PAQ. Os outros artigos apresentam melhorias para metaheurísticas já aplicadas ao PAQ. Estes artigos são muito referenciados na literatura. Além dos artigos serem pioneiros na aplicação de metaheurísticas, e sua grande utilização pela comunidade científica, outro motivo para escolha dos artigos foi a disponibilidade dos códigos-fonte, que são encontrados na biblioteca de referência QAPLIB - *A Quadratic Assignment Problem Library* [11], com exceção do artigo de Drezner [28].

Este capítulo está organizado da seguinte maneira: primeiramente, a Seção 6.1 apresenta a definição geral do Problema de Atribuição Quadrática e uma sucinta revisão bibliográfica, mostrando o ciclo de vida do problema; a Seção 6.2 dá uma breve visão sobre os artigos selecionados, a Seção 6.3 mostra a análise dos quatro artigos

selecionados.

6.1 Problema de Atribuição Quadrática

O Problema de Atribuição Quadrática (PAQ) foi definido inicialmente por Koopmans e Beckmann em 1957 [59], como uma aplicação relacionada à economia. Foi amplamente aplicado em problemas de *layout*, como por exemplo, em planejamento de hospitais e construção de campus universitários. Todavia, existem várias aplicações práticas para esta modelagem, tais como minimizar a quantidade de ligações entre componentes de placas de circuitos eletrônicos; alocação de serviços ou pessoas em postos policiais, supermercados, escolas; escalonamento de horários, análise de reações químicas; computação paralela e distribuída, entre outras aplicações [62].

O problema consiste na alocação de custo mínimo de um conjunto de n atividades a um conjunto de n locais. O objetivo é minimizar o custo associado às distâncias entre os lugares e o fluxo entre as atividades. Dadas 2 matrizes de ordem n , $F = (f_{ij})$ e $D = (d_{kl})$, onde f_{ij} é o fluxo entre as atividades i e j , d_{kl} é a distância entre os locais k e l , e um conjunto N de inteiros, o problema pode ser definido por:

$$\min_{p \in \pi_N} \sum_{i=1}^n \sum_{j=1}^n f_{ij} d_{p(i)p(j)}, \quad (6-1)$$

onde π_N é o conjunto de todas as permutações de N .

Vários *surveys* sobre o assunto foram publicados, nos quais se baseiam esta Seção [16, 29, 62, 78, 87]. O trabalho de Loiola, Abreu e Boaventura-Netto [62] se destaca por referenciar uma extensa quantidade de publicações sobre o PAQ, em relação a modelagens, métodos utilizados, tanto exatos como heurísticos, bem como limites inferiores para o problema. Por isto, as referências a seguir baseiam-se neste *survey*. Em relação às formulações e modelagens, destacam-se:

- **Formulações por Programação Inteira (PLI):** Koopmans e Beckmann, 1957 [59]; Steinberg, 1961; Lawler, 1963; Gavett e Plyter, 1966; Elshafei, 1977; Bazaraa e Sherali, 1979; Bazaraa e Kirca, 1983; Christofides e Benavent, 1989; Bos, 1993; Mans et al., 1995; Liang, 1996, Torki et al., 1996; Tsuchiya et al., 1996, 2001; Ball et al., 1998; Ishii e Sato, 1998; Kaibel 1998; Kochhar et al., 1998; Martin, 1998; Spiliopoulos e Sofianopoulou, 1998; Junger e Kaibel, 2000, 2001; Siu e Chang, 2002; Yu e Sarker, 2003; Fedjki e Duffuaa, 2004;
- **Formulações por Programação Inteira Mista (PLIM):** Lawler, 1963; Love e Wong, 1976; Kaufman e Broeckx, 1978; Bazaraa e Sherali, 1980; Christofides et al., 1980; Burkard e Bonniger, 1983; Frieze e Yadegar, 1983; Assad e Xu, 1985;

Adams e Sherali, 1986; Christofides e Benavent, 1989; Adams e Johnson, 1994; Drezner, 1995; Gouveia e Voß, 1995; Milis e Magirou, 1995; Padberg e Rijal, 1996; White, 1996; Ramachandran e Pekny, 1998; Karisch et al., 1999; Ramakrishnan et al., 2002;

- **Formulação por Permutações:** Hillier e Michael, 1966; Graves e Whinston, 1970; Pierce e Crowston, 1971; Burkard e Stratman, 1978; Roucairol, 1979, 1987; Burkard, 1984; Frenk et al., 1985; Bland e Dawson, 1991, 1994; Battiti e Tecchiolli, 1994; Bui e Moon, 1994; Chakrapani e Skorin-Kapov, 1994; Fleurent e Ferland, 1994; Li et al., 1994; Mautor e Roucairol, 1994; Li e Smith, 1995; Taillard, 1995; Bozer e Suk-Chul, 1996; Colorni et al., 1996; Huntley e Brown, 1996; Peng et al., 1996; Cung et al., 1997; Mavridou e Pardalos, 1997; Merz e Freisleben, 1997; Nissen, 1997; Pardalos et al., 1997; Angel e Zissimopoulos, 1998; Deineko e Woeginger, 1998; Talbi et al., 1998, 2001; Tian et al., 1996, 1999; Tansel e Bilen, 1998; Abreu et al., 1999; Fleurent e Glover, 1999; Gambardella et al., 1999; Maniezzo e Colorni, 1999; Ahuja et al., 2000; Angel e Zissimopoulos, 2000, 2001, 2002; Stutzle e Holger, 2000; Arkin et al., 2001; Pitsoulis et al., 2001; Abreu et al., 2002; Gutin e Yeo, 2002; Hasegawa et al., 2002, Boaventura-Netto, 2003; Rangel e Abreu, 2003;
- **Formulação Traço:** Edwards, 1980; Finke et al., 1987; Hadley et al., 1990, 1992; Hadley, 1994; Karisch e Rendl, 1995; Anstreicher et al., 1999; Anstreicher e Brixius, 2001;
- **Relaxação por Programação Semidefinida (PSD):** Karisch et al., 1994; Zhao et al., 1998; Wolkowicz, 2000;
- **Formulação por grafos:** Yamada, 1992; White, 1995; Abreu et al., 1999; Marins et al., 2004;

Diferentes métodos exatos e heurísticos têm sido utilizados para a resolução do PAQ. Os métodos exatos mais utilizados são baseados em enumeração implícita, programação dinâmica e planos de corte. E os métodos heurísticos e metaheurísticos aplicados ao PAQ são vários, incluindo também a combinação de diferentes métodos. Os primeiros métodos heurísticos utilizados foram os construtivos, montando a solução com permutações. Também foram utilizados métodos enumerativos e de busca local. No início dos anos 80, com a criação do conceito de metaheurística, vários métodos foram amplamente utilizados para o PAQ, tais como *Simulated Annealing*, *GRASP*, Busca Tabu, *Scatter Search*, *Variable Neighbourhood Search*, Algoritmos Genéticos, Colônia de Formigas, Redes Neurais, Algoritmos Meméticos, Algoritmos Transgenéticos e metaheurísticas híbridas¹, como combinação de *Simulated Annealing* com Algoritmos

¹Ver definição de metaheurística híbrida no final da Seção 2.3, Capítulo 2.

Genéticos, *Simulated Annealing* com Busca Tabu, Busca Tabu com Redes Neurais. Conforme levantamento realizado por Loiola, Abreu e Boaventura-Netto [62], a seguir estão algumas referências de métodos desenvolvidos para a PAQ:

- **Métodos exatos:** Gilmore, 1962; Land, 1963; Lawler, 1963; Gavett e Plyter, 1966; Nugent et al., 1968; Graves e Whinston, 1970; Pierce e Crowston, 1971; Burkard e Stratman, 1978; Kaufman e Broeckx, 1978; Bazaraa e Elshafei, 1979; Mirchandani e Obata, 1979; Roucairol, 1979; Bazaraa e Sherali, 1980; Burkard e Derigs, 1980; Edwards, 1980; Bazaraa e Kirca, 1983; Burkard e Bonniger, 1983; Kaku e Thompson, 1986; Roucairol, 1987; Christofides e Benavent, 1989; Pardalos e Crouse, 1989; Burkard, 1991; Padberg e Rinaldi, 1991; Laursen, 1993; Mautor e Roucairol, 1994; Mans et al., 1995; Bozer e Suk-Chul, 1996; Clausen e Perregaard, 1997; Pardalos et al., 1997; Brungger et al., 1998; Ball et al., 1998; Urban, 1998; Spiliopoulos e Sofianopoulou, 1998; Brixius e Anstreicher, 2001; Hahn et al., 2001; Miranda et al., 2005;
- **Heurísticas:**
 - **Métodos construtivos:** Armour e Buffa, 1963; Buffa et al., 1964; Burkard, 1991; Sarker et al., 1995, 1998; Misevicius, 1997; Tansel e Bilen, 1998; Fleurent e Glover, 1999; Misevicius e Riskus, 1999; Arkin et al., 2001; Gutin e Yeo, 2002; Yu e Sarker, 2003;
 - **Métodos enumerativos:** Burkard e Bonniger, 1983; West, 1983; Nissen e Paul, 1995;
 - **Métodos de melhoria:** Heider, 1973; Mirchandani e Obata, 1979; Bruijs, 1984; Pardalos et al., 1993; Burkard e Cela, 1995; Li e Smith, 1995; Anderson, 1996; Talbi et al., 1998; Deineko e Woeginger, 2000; Misevicius, 2000; Mills et al., 2003;
- **Metaheurísticas:**
 - ***Simulated Annealing*:** Burkard e Rendl, 1984 [12]; Wilhelm e Ward, 1987; Connolly, 1990; Bos, 1993; Yip e Pao, 1994; Burkard e Cela, 1995; Peng et al., 1996; Tian et al., 1996, 1999; Mavridou e Pardalos, 1997; Chiang e Chiang, 1998; Later, Abreu et al., 1999; Misevicius, 2000, 2003; Tsuchiya et al., 2001; Siu e Chang, 2002; Baykasoglu, 2004;
 - **Algoritmos Genéticos:** Davis, 1987; Goldberg, 1989; Bui e Moon, 1994; Tate e Smith, 1995; Mavridou e Pardalos, 1997; Kochhar et al., 1998; Tavakkoli-Moghaddain e Shayan, 1998; Gong et al., 1999; Drezner e Marcoulides, 2003; El-Baz, 2004; Wang e Okazaki, 2005; Drezner, 2005;
 - **Colônia de Formigas:** Maniezzo e Colorni, 1995, 1999; Colorni et al., 1996; Dorigo et al., 1996; Gambardella et al., 1999; Stutzle e Dorigo, 1999; Stutzle

- e Holger, 2000; Talbi et al., 2001; Middendorf et al., 2002; Solimanpur et al., 2004; Randall, 2004; Ying e Liao, 2004; Acan, 2005;
- **Busca Tabu:** Skorin-Kapov, 1990, 1994; Bland e Dawson, 1991; Taillard, 1991 [105]; Rogger et al., 1992; Chakrapani e Skorin-Kapov, 1993; Battiti e Tecchiolli, 1994; Misevicius, 2003, 2005; Drezner, 2005;
 - **GRASP:** Li et al., 1994; Feo e Resende, 1995; Resende, Pardalos e Li, 1996 [96]; Pardalos, Pistoulis e Resende, 1997 [86]; Fleurent e Glover, 1999; Ahuja et al., 2000; Rangel et al., 2000; Pitsoulis et al., 2001; Oliveira et al., 2004;
 - **Variable Neighborhood Search:** Mladenovic e Hansen, 1997; Taillard e Gambardella, 1999;
 - **Metaheurísticas Híbridas:** Bolte e Thonemann, 1996; Battiti e Tecchiolli, 1994; Bland e Dawson, 1994; Chiang e Chiang, 1998; Talbi et al., 1998; Misevicius, 2001, 2004; Hasegawa et al., 2002; Youssef, et al., 2003; Fleurent e Ferland, 1994; Ahuja et al., 2000; Lim et al., 2000, 2002; Drezner, 2003 [28]; Balakrishnan et al., 2003; Misevicius, 2004; Dunker et al., 2004;

No estudo de Loiola, Abreu e Boaventura Neto [62] sobre o PAQ é mostrado que dentre os métodos mais utilizados, os metaheurísticos foram os mais aplicados ao problema, praticamente o dobro de trabalhos encontrados em relação aos métodos exatos. Em relação aos métodos metaheurísticos, os métodos híbridos foram os mais utilizados. Os métodos heurísticos puros que foram mais explorados sobre o PAQ foram os métodos *Simulated Annealing* e GRASP.

6.2 Artigos Seleccionados

O artigo de Burkard e Rendl, de 1984 [12] (Vide Tabela 6.1), apresenta a aplicação do método *Simulated Annealing*², utilizando a modelagem de Koopmans-Beckmann para o PAQ [59]. Este trabalho foi o primeiro a utilizar esta metaheurística para o PAQ. São utilizadas instâncias de referência, encontradas na QAPLIB, propostas por Nugent, 1968; Krarup, 1972 e Steingerb, 1961 apud [12]. O algoritmo é avaliado usando duas métricas, tempo e qualidade da solução. Para comparar o tempo computacional, o algoritmo proposto é comparado com o trabalho de Burkard e Derigs, de 1980 apud [12], que consiste numa heurística baseada no método Monte Carlo para o PAQ. Já em relação à qualidade da solução, o algoritmo é comparado com o trabalho de Burkard e Bönniger, de 1983 apud [12]. Os resultados encontrados apresentam uma diferença de 1% a 2% em relação às soluções ótimas conhecidas.

²Para descrição do método *Simulated Annealing*, Ver Seção 2.3.2.

Tabela 6.1: Artigos seleccionados para análise.

N.º	Título	Autor(es)	Ano
1	<i>A thermodynamically motivated simulation procedure for combinatorial optimization problems</i> [12]	Burkard e Rendl	1984
2	<i>Robust Taboo search for the quadratic assignment problem</i> [105]	Taillard	1991
3	<i>Algorithm 769: Fortran Subroutines for Approximate Solution of Sparse Quadratic Assignment Problems Using GRASP</i> [86]	Pardalos, Pitsoulis e Resende	1997
4	<i>A New Genetic Algorithm for the Quadratic Assignment Problem</i> [28]	Drezner	2003

O trabalho de Taillard, de 1991 [105], mostra um algoritmo de Busca Tabu³ para o PAQ. Este trabalho cita o artigo anterior, de Burkard e Rendl [12]. As instâncias utilizadas são as mesmas do artigo 1, mais as de Skorin-Kapov (1990), e Wilhelm a Ward⁴ apud [105], com tamanho variando de 15 a 100. Além de apresentar um algoritmo gerador de instâncias aleatórias, o trabalho compara a abordagem proposta com vários trabalhos, em relação à qualidade da solução: dois trabalhos que utilizaram o método *Simulated Annealing*, de Burkard e Rendl [12] e Connolly, 1990 apud [105]; outros dois trabalhos com a técnica Busca Tabu de Skorin-Kapov, 1990 apud [105]; e outro utilizando a técnica de Máquina de Boltzman, de Chakrapani e Skorin-Kapov, 1990 apud [105]. O algoritmo proposto obteve soluções melhores em relação aos outros, para instâncias de tamanho 42 a 100, sendo que as instâncias de tamanho 15 a 36 são encontrados os mesmos resultados. São implementados dois métodos de modo paralelo, e é mostrado que soluções de qualidade são obtidas quando o número de processadores utilizados é proporcional ao tamanho do problema.

O terceiro artigo analisado, de Pardalos, Pitsoulis e Resende, de 1997 [86], utiliza o método GRASP⁵ para o PAQ Esperso. Diferente dos dois artigos anteriores, a modelagem do problema é baseada em matrizes esparsas. O modelo utilizado é o de Koopmans e Beckmann, entretanto é feito um cálculo para saber se pelo menos uma matriz de entrada é esparsa. Foram testadas instâncias presentes na QAPLIB, de tamanho 12 a 100. O algoritmo é comparado com o mesmo método, GRASP, mas para matrizes densas, pelos mesmo autores [96]. O algoritmo encontra soluções 35% mais rápido que o outro, sem afetar a qualidade da solução, e para matrizes com esparsidade maior ou igual a 0,8, isto é, a quantidade de elementos na matriz iguais a zero é maior ou igual a 80%, o algoritmo chega a ser 300% mais rápido.

³Para descrição do método Busca Tabu, Ver Seção 2.3.3.

⁴As instâncias destes autores não foram referenciadas.

⁵Para descrição do método GRASP, Ver Seção 2.3.4.

Por fim, o trabalho de Drezner, de 2003 [28], apresenta uma variação da metaheurística Algoritmos Genéticos para o PAQ. A modelagem aplicada é a que foi utilizada em todos os artigos anteriores, de Koopmans-Beckmann. A grande diferença no novo método é o mecanismo de reprodução⁶, parte principal do Algoritmo Genético, aliado a uma nova variação do algoritmo de Busca Tabu chamado de Busca Tabu Concêntrico. O algoritmo de busca tabu é aliado ao processo de reprodução como forma de melhorar e refinar a população, para produzir melhores soluções. São desenvolvidas três variações do algoritmo e analisados o desempenho de cada um, comparando também com o trabalho de Ahuja, Orlin e Tiwari, de 2000 apud [28]. O método desenvolvido encontra boas soluções em pouco tempo de processamento, sendo que em relação ao de Ahuja, Orlin e Tiwari, foi encontrado um desempenho 20 vezes melhor, tanto em relação ao tempo quanto à qualidade da solução.

6.3 Análise dos Artigos

Esta avaliação dos artigos foi feita com base nos itens descritos nos Capítulos 3 a 5 e sumarizadas no *checklist* desenvolvido (Seção 5.2). Portanto, foi dividida em: revisão da literatura (Subseção 6.3.1), modelo experimental (Subseção 6.3.2), apresentação dos algoritmos (Subseção 6.3.3), implementação (Subseção 6.3.4), relato dos resultados (Subseção 6.3.5) e conclusões (Subseção 6.3.6). Em cada item, por exemplo, a apresentação dos algoritmos, são descritas as análises dos artigos da Tabela 6.1, de forma a obter uma comparação entre eles, citando quais as recomendações, apresentadas nos Capítulos 3, 4 e 5 foram seguidas e as potenciais falhas encontradas.

6.3.1 Revisão da Literatura

Como foi visto na Seção 3.3, a revisão da literatura é a base para o estudo de um problema, pois com o domínio do problema, é possível definir e delimitar claramente os objetivos para o estudo de um problema e de um experimento computacional. Entre as informações que devem ser levantadas, estão saber se o problema ou o algoritmo são novos, quais são os trabalhos relacionados, e quais são as modelagens e algoritmos foram desenvolvidos.

No artigo 1 da Tabela 6.1, de Burkard e Rendl [12], é citada e aplicada a modelagem de Koopmans-Beckmann. O algoritmo apresentado para resolução de problemas de otimização é o *Simulated Annealing*, que já tinha sido criado por Metropolis et al.[70], entretanto, para a resolução do PAQ ele ainda não tinha sido testado. O artigo não descreve

⁶Para descrição do método Algoritmos Genéticos, Ver Seção 2.3.5.

os trabalhos relacionados, somente cita dois trabalhos Burkard e Derigs, 1980 e Burkard, 1983 apud [12], pois são utilizados para comparação dos resultados. O PAQ na época da publicação do artigo já tinha aproximadamente 20 anos de estudo, não é considerado se o problema é novo, somente é ressaltada sua grande aplicabilidade.

No artigo 2, de Taillard [105] (Tabela 6.1), é declarado que o problema não é novo, pois tinha sido estudado por aproximadamente 30 anos. O problema é definido, utilizando o mesmo modelo que o artigo 1. O método Busca Tabu já tinha sido estudado para a resolução do PAQ, entretanto, o algoritmo pôde ser considerado novo, pois é uma variação da Busca Tabu já que sua implementação é feita em paralelo. São citados os métodos que foram desenvolvidos, contudo o *Simulated Annealing* e Busca Tabu não são explicados. Algo interessante é a citação de uma instância encontrada na literatura que ainda não tinha sido encontrada nenhuma solução para a mesma, a de Steinberg, 1961 apud [105], de tamanho 36.

O artigo 3, de Pardalos, Pitsoulis e Resende [86] (Tabela 6.1), mostra que o PAQ não é novo, pois apresenta a data em que foi definido (1957). O modelo estudado é baseado no modelo de Koopmans-Beckmann, com uma variação para ter como entrada matrizes esparsas. O método GRASP já tinha sido aplicado ao PAQ para matrizes densas [96], mas não para matrizes esparsas, tornando o algoritmo proposto novo. Foram dadas referências para estudo do PAQ e GRASP.

O artigo 4, de Drezner [28] (Tabela 6.1), apesar de utilizar o modelo de Koopmans-Beckmann, como todos os anteriores, não mostra a referência ao autor. São citados alguns trabalhos relacionados, inclusive os artigos aqui estudados, o de Burkard e Rendl [12] e Taillard [105]. Os algoritmos propostos são descritos como novos, tanto o Algoritmo Genético quanto a Busca Tabu e comparados com o trabalho de [1].

Este primeiro passo a ser feito, a revisão bibliográfica, presente na Seção 3.3, está resumida Tabela 6.2, que equivale à primeira parte do *checklist* para relato de experimento computacional, definido na Tabela 5.2. Desta forma, os itens cobertos pelos artigos foram:

A - Revisão da Literatura				
Itens Recomendados	Artigo 1	Artigo 2	Artigo 3	Artigo 4
1) O problema é novo?				
2) O algoritmo proposto é novo?	×	×	×	×
3) O algoritmo já foi implementado?				
4) O algoritmo já foi estudado para o problema em questão?				
5) Fala sobre modelagens existentes?			×	×
6) Fala sobre métodos já desenvolvidos para o problema?	×	×	×	×
7) Definição clara do problema.	×	×	×	×

Tabela 6.2: Itens cobertos sobre a Revisão da Literatura

6.3.2 Modelo Experimental

O modelo experimental influencia toda a condução do experimento. Primeiramente, os objetivos levantados no início do experimento devem ser claros, para ao final, ser possível obter as conclusões esperadas. A Seção 3.4 mostra vários possíveis objetivos que podem ser seguidos em uma pesquisa sobre algoritmos. O modelo experimental também deve ser definido, e a Seção 4.1 mostra os possíveis modelos experimentais que podem ser utilizados. Estes modelos são resumidos em modelo experimental básico, que trabalha com comparações de instâncias e algoritmos em sua maneira mais simples, e o modelo experimental estatístico, que oferece vários métodos e um rigor estatístico para análise de dados.

Outro fator importante é a descrição dos itens e parâmetros utilizados na execução do experimento. Devem ser relatados a quantidade de testes feitos com cada instância, quais e quantas sementes foram utilizadas (caso seja utilizada aleatoriedade), quantas iterações o algoritmo executou e qual critério de parada foi utilizado. Interessante notar que estes parâmetros podem variar. Por exemplo, pode-se utilizar vários critérios de parada em um algoritmo, por isso o relato da variação destes parâmetros é essencial para permitir a reprodução do experimento (Seção 5.2.3).

Em relação às instâncias, deve-se descrever o conjunto de teste utilizado, informando se são utilizadas instâncias reais ou aleatórias e se são instâncias de referência, para permitir a comparação de um experimento com outro. Assim, outro pesquisador pode testar um novo algoritmo com o mesmo conjunto de instâncias. Se forem criadas novas instâncias, o algoritmo gerador de instâncias deve ser descrito (Seção 4.2).

Vistos os pontos importantes do modelo experimental, segue a análise dos artigos da Tabela 6.1. No artigo 1, de Burkard e Rendl [12], o objetivo do experimento é comparar uma abordagem com técnicas já implementadas, ou seja, utilizar a metaheurística *Simulated Annealing* para resolver o modelo de Koopmans-Beckmann para o PAQ; testar e comparar com dois outros trabalhos (Seção 4, p. 171). O modelo experimental utilizado é o básico (instâncias $\times$ algoritmos), onde são comparados tempo computacional e qualidade da solução. O experimento não é descrito, isto é, dados como a quantidade de réplicas (testes repetidos) feitos para cada instância e quantidade máxima de iterações, não são citadas. É citado o critério de parada, que corresponde a uma certa quantidade de iterações, entretanto não é definido quantas iterações o método executa. Neste trabalho não são geradas novas instâncias, logo o item B5 Tabela 5.2, não é necessário e o conjunto de instâncias de teste utilizado é definido e referenciado (Tabela 1, p. 171).

O objetivo do trabalho desenvolvido no artigo 2, de Taillard [105] (Tabela 6.1), consiste em testar e melhorar algoritmos, e também comparar uma abordagem com técnicas já implementadas, abrangendo os dois objetivos (a) e (b) do item B1, definidos na Tabela 5.2. Desta forma, os objetivos do experimento são: propôr um método baseado

em Busca Tabu mais robusto, que utiliza menos parâmetros, mais fácil de implementar e capaz de obter boas soluções, e então comparar o algoritmo proposto com outros trabalhos, e também com variações dos parâmetros do próprio algoritmo, para avaliar a qualidade da solução.

Como no artigo 1, o artigo 2 também utiliza o modelo experimental básico, em que são comparados tempo computacional e qualidade da solução. O experimento é descrito em partes soltas no texto, dificultando a compreensão da condução e execução deste. Um dado importante presente neste artigo, que não foi citado no artigo 1, é quantas vezes são executados os testes, para obter a média. Neste trabalho, são feitas 30 repetições de testes para cada instância de referência, e para as instâncias geradas aleatoriamente, cada teste inicia com uma solução diferente, sendo que este conjunto de soluções iniciais é o mesmo para todos os métodos heurísticos, e cada instância é testada 300 vezes para encontrar um solução boa (Seção 6, p. 451). As outras instâncias, que são de referência, são testadas 30 vezes cada uma, aproximadamente. As sementes utilizadas são descritas. O critério de parada utilizado é baseado na quantidade de iterações que o método executa. A quantidade mínima de iterações é dada por um número proporcional a N , sendo N o tamanho da instância, e são realizadas 1000, $4N$ e N^2 iterações.

O artigo 2 propõe um gerador aleatório de instâncias, que é descrito sucintamente. Entretanto, não é documentado plenamente e não é disponibilizado acesso público para outros pesquisadores. O conjunto de instâncias de teste utilizado não está explicado, todavia, as instâncias são conhecidas e estão nomeadas, pois pertencem à QAPLIB (Seção 3, p. 443).

No artigo 3, de Pardalos, Pitsoulis e Resende [86] (Tabela 6.1), os objetivos também abrangem os tópicos (a) e (b) do item B1, definidos na Tabela 5.2, ou seja, consistem em testar e melhorar algoritmos, e também comparar uma abordagem com técnicas já implementadas. Em suma, o objetivo é tentar resolver o Problema de Atribuição Quadrática Esperso. A justificativa da utilização desta classe de problemas é a aplicabilidade no mundo real, como problemas de análise de dados e escalonamentos de processos. Claramente, é explicitado que o algoritmo proposto é comparado com algoritmos gerais, isto é, são comparados os métodos GRASP para PAQ Esperso e GRASP para PAQ Denso, com um vasto conjunto de instâncias.

Igualmente como os artigos anteriores, o experimento baseia-se no modelo experimental básico, para analisar qualidade de solução e tempo computacional. O diferencial neste artigo é a descrição organizada de como é realizado o experimento, encontrada na Seção 6, p. 207: para cada instância, são feitas 40 repetições de testes, com iterações no algoritmo que variaram de 16 a 2048 iterações. São utilizadas cinco sementes para geração aleatória. O ambiente computacional é citado, incluindo as *flags* do compilador. Também é citada a fonte das instâncias, a QAPLIB, a dimensão do conjunto

de instâncias testado, e que pelo menos uma matriz era simétrica, ou a de fluxos ou a de distâncias. Estes itens são importantes para a reprodução do experimento. Igualmente ao artigo 1, não são geradas novas instâncias, portanto o item B5 da Tabela 5.2 não é necessário. O conjunto de instâncias de teste é enorme, sendo que cada instância tem seu nome indicando o tamanho e autor, que pode ser visto nas tabelas de resultados.

O artigo 4, de Drezner [28] (Tabela 6.1), os objetivos também abrangem os tópicos (a) e (b) do item B1, definidos na Tabela 5.2, ou seja, consistem em testar e melhorar algoritmos, e também comparar uma abordagem com técnicas já implementadas. O objetivo é tentar resolver o PAQ com variações na metaheurística de Algoritmos Genéticos, e comparar com outros trabalhos que implementam a mesma metaheurística.

Assim como em todos os artigos, o experimento baseia-se no modelo experimental básico, para analisar qualidade de solução e tempo computacional. Este artigo possui uma Seção sobre os experimentos computacionais (Seção 3, p. 325), são feitos testes repetidos por instância, que variaram de 20, 100 e 200 vezes, sendo que são testadas 29 instâncias, as matrizes de entrada são simétricas e o tamanho das instâncias varia de 30 a 100, não são conhecidas soluções ótimas. O critério de parada utilizado é a quantidade de iterações, que é gerada aleatoriamente, para executar o procedimento principal do algoritmo. As instâncias testadas são descritas e estão na QAPLIB.

O modelo experimental utilizado em todos os artigos foi o modelo experimental básico (instâncias $\times$ algoritmos). Em geral, os quatro artigos apresentaram novos algoritmos, ou variações de algoritmos já desenvolvidos, e todos fizeram comparações com o objetivo de verificar a acurácia dos novos métodos. Interessante notar que, como todos os trabalhos tinham praticamente os mesmos objetivos, era de se esperar que abrangessem a maioria dos itens recomendados no *checklist* da Tabela 5.2. Os requisitos de relato do modelo experimental são representados na Tabela 6.3:

B - Modelo Experimental					
Itens Recomendados		Artigo 1	Artigo 2	Artigo 3	Artigo 4
1) Definição clara dos objetivos do experimento	a) Comparar uma abordagem com técnicas já implementadas	×	×	×	×
	b) Testar e melhorar algoritmos para problemas difíceis	×	×	×	×
	c) Comparar algoritmos existentes e estruturas de dados para problemas				
	d) Comprovar e refinar conjecturas				
	e) Desenvolver bibliotecas para algoritmos básicos e estruturas de dados				
	f) Desenvolver ferramentas para facilitar o projeto e análise de algoritmos				
2) Modelo experimental	a) Básico (instâncias × algoritmos)	×	×	×	×
	b) Estatístico				
	c) Bloqueio de instâncias				
	d) Balanceamento de qualidade e tempo				
3) Descrição da execução do experimento	a) Quantidade de testes repetidos feitos por instância		×	×	×
	b) Quantidade de instâncias testadas				×
	c) Quantidade de sementes utilizadas		×	×	
	d) Quantidade máxima de iterações		×	×	×
	e) Critério de parada utilizado	×	×	×	×
4) Descrição do conjunto de instâncias de teste utilizado	a) Instâncias reais ou aleatórias	×	×	×	×
	b) Se são instâncias de referência	×	×	×	×
5) Geração de novas instâncias	a) Descrição do gerador de instâncias	-	×	-	-

Tabela 6.3: *Itens cobertos sobre o Modelo Experimental.*

Como nos artigos 1, 2 e 4 não foram criadas novas instâncias, os itens 4(a) e 4(b) da Tabela 6.3 não são necessários. Portanto, estes itens foram desconsiderados para os respectivos artigos e marcados com um traço (-) na Tabela.

6.3.3 Apresentação dos Algoritmos

Sobre a apresentação dos algoritmos dois itens são essenciais, uma descrição sobre a classe do problema que o algoritmo resolve e a descrição completa do algoritmo. Em relação à classe do problema, deve-se deixar claro para qual tipo de instâncias o algoritmo encontra soluções, e também até que tamanho de instância o algoritmo encontra boas soluções. Já para a descrição do algoritmo, deve-se descrever a estratégia inicial, quais foram os critérios de inicialização e de parada, como foram modelados os parâmetros do algoritmo, como e quais foram utilizados.

Pode-se apresentar análises, para melhor compreender o comportamento do algoritmo. Uma delas é analisar a quantidade de trabalho feita pelo algoritmo por iteração,

ou por fase, podendo mostrar o tempo computacional gasto, bem como valores de determinadas variáveis. Apesar da dificuldade de se fazer a análise da complexidade de algoritmos complexos, esta pode ser apresentada, e também pode ser somente de alguma parte do algoritmo. Uma descrição mais completa desses elementos é encontrada na Seção 5.2.3.

O artigo 1 da Tabela 6.1, de Burkard e Rendl [12], dá uma explicação sobre a metaheurística utilizada, *Simulated Annealing*, e também do algoritmo utilizado, incluindo o tipo das variáveis, estas são comentadas, como por exemplo, citando que o valor da variável *rep* influencia na qualidade da solução encontrada, visto que esta variável corresponde à quantidade de repetições que o método executa. Entretanto, não é explicitada até que dimensão do problema o algoritmo proposto resolve. Mas pode-se deduzir a dimensão, já que são encontradas soluções para instâncias de teste de tamanho 12 a 36 (Seção 3, p. 170 e 171). Também é dada uma análise da complexidade da movimentação na vizinhança (Seção 4, p. 171).

Na introdução do artigo 2 da Tabela 6.1, de Taillard [105], já é especificado qual tamanho de instância que o algoritmo resolve com eficiência, isto é, instâncias de tamanho até 64 (Seção 1, p. 443). Contudo, são encontradas soluções para instâncias de tamanho máximo 100. É apresentada uma análise da complexidade da movimentação na vizinhança, como no artigo 1 (Seção 3.1, p. 445 e 446). Um gerador de instâncias aleatórias é proposto, incluindo algoritmo com explicações (Seção 4, p. 449). O algoritmo principal, com a técnica Busca Tabu, não é exposto. Entretanto, a Seção 3, p. 445, é dedicada a explicar os parâmetros do método, sobre como são feitos os movimentos na vizinhança (Seção 3.1), sobre a lista tabu (Seção 3.2), sobre a função de aspiração (Seção 3.3) e sobre o cálculo do tamanho da lista tabu (Seção 3.4). Também há uma descrição sobre a paralelização (p. 450). Por fim, é mostrada a complexidade do algoritmo da função objetivo, sobre o algoritmo de movimentação, e sobre as iterações do método, pois variam de acordo com N , o tamanho da instância (Seção 6, p. 451).

O artigo 3 da Tabela 6.1, de Pardalos, Pitsoulis e Resende [86], define a classe de problema a ser trabalhado, o Problema de Atribuição Quadrática Esparsa. O tamanho das instâncias testadas variam de 12 a 100 (Seção 1, p. 196). As Seções 2 e 3, p. 197, são dedicadas à explicação dos algoritmos desenvolvidos. O algoritmo proposto é o GRASP, com uma explicação completa sobre as duas fases, a de construção, que seria a estratégia inicial de GRASP, e a de busca, incluindo a descrição de todos os parâmetros. Há melhor legibilidade na explicação, pois os algoritmos descritos possuem numeração nas linhas, facilitando a explicação, diferente dos artigos 1 e 2, que não possuem.

O artigo 4 de Drezner [28] (Tabela 6.1), explica os algoritmos propostos e apresenta-os em forma de itens/passos a seguir (Seção 2, p. 320). Como a função principal de um algoritmo genético é a reprodução, são apresentadas duas estratégias para o PAQ,

chamadas de *cohesive merging procedure* e *scrambled merging procedure*. A estratégia inicial é explicada, onde é gerada uma população de forma aleatória e depois refinada com o algoritmo de busca tabu. Um procedimento adicional é usado para refinar a população gerada, chamado de *concentric tabu search*, uma variação da busca tabu. Alguns parâmetros são descritos, tais como o tamanho da lista tabu e o tamanho da população.

O passo de apresentação dos algoritmos, descrito na Seção 5.2.3, é sumarizado na terceira parte do *checklist* da Tabela 5.2. Em síntese, os quatro artigos atenderam quase totalmente os requisitos, dispostos na Tabela 6.4:

C - Apresentação dos Algoritmos					
Itens Recomendados		Artigo 1	Artigo 2	Artigo 3	Artigo 4
1) Descrição completa do algoritmo		×	×	×	×
2) Classe do problema que o algoritmo proposto resolve	a) Qual tipo de instâncias o algoritmo encontra soluções?	×	×	×	×
	b) O algoritmo encontra soluções para instâncias de até que tamanho?	×	×	×	×
3) Descrição da técnica de estratégia inicial		×	×	×	×
4) Uso de diferentes critérios de inicialização			×	×	
5) Uso de diferentes critérios de término		×	×	×	
6) Dados dos parâmetros da heurística (Ex: tamanho da lista tabu)		×	×	×	×
7) Uso de diferentes valores nos parâmetros do algoritmo			×	×	×
8) Análise de complexidade do algoritmo		×	×		
9) Análise da quantidade de trabalho por iteração			×		

Tabela 6.4: Itens cobertos na Apresentação dos Algoritmos.

6.3.4 Implementação

Como descrito na Seção 5.2.3, a descrição da implementação é importante para permitir a reprodução do experimento, pois se os detalhes forem bem explicitados, é possível fazer uma implementação muito parecida se o código-fonte não está disponível, e então fazer comparações. Os itens fundamentais que devem ser citados num relato de experimento são: a linguagem de programação, compilador e opções, ambiente computacional, descrição dos dados de entrada, descrição das configurações, condições em que o código está disponível, descrição das técnicas de pré-processamento entre outras, disponibilidade do código-fonte, e se possível um manual de instruções.

O artigo 1, de Burkard e Rendl [12] (Tabela 6.1), relata poucos itens sobre implementação. Primeiramente, não é citada a linguagem de programação nem compilador utilizados. Todavia, foi possível saber a linguagem utilizada (Fortran), após o *download* do código-fonte na repositório QAPLIB [11]. O artigo apresenta uma prévia visão do ambiente computacional, citando o modelo do computador, um Univac 1100/81, mas não cita o sistema operacional (Seção 4, p. 171).

Os dados de entrada da implementação não são citados, mas pôde-se deduzir por um comentário no código, e também, por a maioria dos códigos-fontes no repositório QAPLIB ter um padrão de entrada, que é a dimensão das matrizes, e depois as duas matrizes, sendo a primeira de fluxos e a segunda de distâncias. Apesar do código estar no QAPLIB, não é citado no artigo que ele está disponível. Enfim, não são descritas técnicas de pré-processamento, estratégia inicial e de implementação.

O artigo 2 de Taillard [105] (Tabela 6.1) é o que mais deixa a desejar sobre dados da implementação. O código-fonte desenvolvido em C++ também foi encontrado no QAPLIB. No relato não existem dados sobre linguagem de programação utilizada, compilador, ambiente computacional, descrição dos dados de entrada e outras configurações ou onde o código está disponível. Sobre o ambiente computacional, é citado somente o processador Transputer T800C-G20S (Seção 6, p. 453).

O artigo 3 de Pardalos, Pitsoulis e Resende [86] (Tabela 6.1) é o mais completo, incluindo praticamente todos os itens recomendados. No relato, é dedicada uma seção para o projeto e implementação dos algoritmos (Seção 4, p. 202). É citada a linguagem Fortran 77, e o ambiente computacional utilizado, Unix. O compilador é citado, o f77, incluindo as *flags* utilizadas (Seção 6, p. 207). No trabalho não consta se o código-fonte estava disponível.

Os dados de entrada são descritos juntamente com as subrotinas. Ainda apresentam como exemplo uma figura para eliminar qualquer dúvida. Os dois estágios de GRASP são explicados passo a passo e o que cada variável representa. É mostrada a saída com as iterações de GRASP, o custo das permutações encontrado, a melhor permutação encontrada, entre outros dados de saída. A Seção 5 (p. 207) mostra como compilar os códigos-fonte, deixando bem claro que o programa só encontra soluções para instâncias de tamanho menor ou igual que 256.

No trabalho de Drezner [28] (Tabela 6.1), é utilizado o programa *Microsoft PowerStation Fortran 4.0*, e com isso conclue-se que o algoritmo é implementado em Fortran, mas não sabe-se qual a versão de Fortran, pois este programa suporta as versões 66, 77 e 90 de Fortran, nem as opções do compilador (Seção 3, p. 325). Sobre o ambiente computacional são citados somente o modelo (Toshiba Portege 7200 600Mhz) e processador (Pentium III), não especificando o tamanho da memória. Não são dados muitos detalhes de implementação, como a descrição das configurações, estruturas de

dados ou outras técnicas, dos dados de entrada, e também não informa se o código está disponível.

A Seção 5.2.3 aborda sobre os itens necessários para o relato da implementação, que estão no *checklist* da Tabela 5.2. Enfim, os quatro artigos atenderam poucos requisitos, com exceção do artigo 3. Os itens cobertos estão dispostos na Tabela 6.5:

D - Implementação				
Itens Recomendados	Artigo 1	Artigo 2	Artigo 3	Artigo 4
1) Linguagem de programação			×	
2) Descrição dos dados de entrada			×	
3) Descrição das configurações			×	
4) Descrição de técnicas de pré-processamento			×	
5) Armazenamento dos requisitos e estruturas de dados			×	
6) Compilador			×	×
7) Opções do compilador			×	
8) Sistema Operacional			×	×
9) Hardware (modelo do comp., processador e memória)	×	×	×	×
10) Se o código está disponível				
11) Instruções para uso			×	

Tabela 6.5: *Itens cobertos sobre a Implementação dos Algoritmos.*

6.3.5 Relato dos Resultados

O relato dos resultados, como foi visto no Capítulo 5, é essencial que seja bem detalhado, pois geralmente um experimento computacional possui vários fatores a serem explorados, e estes influem de maneira significativa nos resultados, na comparação e reprodução do experimento. As principais medidas a serem relatadas são em relação às medidas de desempenho, cálculos estatísticos básicos e análise estatística. Também é importante representar os valores encontrados com o uso de gráficos e tabelas, para melhor compreensão dos dados. As medidas de desempenho, encontradas na Seção 3.5, incluem medidas sobre qualidade da solução, esforço computacional e robustez, sendo que estas medidas devem ser justificadas. Os cálculos estatísticos básicos como médias, totais, desvio padrão e erro amostral, podem ser utilizados. Se for utilizada uma análise estatística rigorosa, valores como intervalo de confiança, margem de erro, e dados sobre a amostra são essenciais. A descrição sobre estes itens é mostrada na Seção 5.1, 5.2.1 e no Apêndice A, na Seção A.3.

No artigo 1, de Burkard e Rendl [12] (Tabela 6.1), as medidas de desempenho utilizadas foram a qualidade da solução (Tabela 2, p. 172), mostrando o valor da solução encontrada; e tempo gasto (Tabela 1, p. 171; Tabela 3, p. 173). A robustez é mostrada, pois

o algoritmo conseguiu ter um desempenho satisfatório para todo o conjunto de instâncias de teste utilizado (p. 173,174). Os resultados são expostos de maneira simples e objetiva, em tabelas, utilizando o modelo experimental básico (ver Seção 4.1). Em uma tabela são comparadas a sensibilidade das soluções com relação a soluções iniciais obtidas com mais outros dois trabalhos, com a melhor, a pior e a solução média obtidas. Não há explicação de quantos testes foram feitos para cada instância, ou seja, não há como saber como foram encontrados estes valores (Tabela 2, p. 172).

Há também comparações de tempo de execução, especificando que a metaheurística é testada com 10 inícios. São mostrados os parâmetros de controle utilizados. Os algoritmos são testados com os mesmos valores nos parâmetros (Tabela 2, p. 171, Seção 4, p. 172). Depois de verificar estes dois itens, é analisado o tempo juntamente com a qualidade da solução, e com isso é possível constatar que o algoritmo proposto obteve melhores resultados. Não são utilizados gráficos.

No artigo 2 de Taillard [105] (Tabela 6.1), são explicadas as duas abordagens utilizadas para comparar e analisar algoritmos: primeiro, é feita uma escolha arbitrária de uma solução; segundo, o algoritmo pára quando o primeiro ótimo local é encontrado, e depois o método Busca Tabu proposto é testado, com quantidade de iterações definidas para mil (1000) e $4N$, sendo N o tamanho da instância. Esta quantidade foi estipulada para analisar se a quantidade de iterações cresce mais rápido que o tamanho da instância. São dadas a porcentagem média das melhores soluções conhecidas dos métodos heurísticos comparados. Ainda no artigo 2, na Tabela 3, p. 453, é dada uma maior precisão nos resultados, isto é, os resultados passam a ter três casas decimais após a vírgula. Os dados apresentados são mais completos: média, desvio padrão, tamanho mínimo e máximo da lista tabu, e um valor da função de aspiração do método.

Em relação a gráficos, o artigo 2 apresenta quatro, comparando as soluções obtidas quando varia-se aleatoriamente o tamanho s da lista tabu, entre s_{min} e s_{max} , sendo que o tamanho inicial começa com duas vezes o tamanho da instância (Fig. 1, p. 448). Outro gráfico mostra, para uma determinada instância, qual a quantidade média de iterações necessárias para chegar a uma solução próxima de uma solução ótima. Entretanto, o comportamento heurístico não permite fazer uma análise mais confiável, pois cada instância pode mostrar um comportamento diferente (Fig. 2, p. 448). É também mostrada a eficiência da paralelização na Fig. 3, p. 450. O cálculo da eficiência é dado pela razão do tempo de CPU teórico pelo tempo de CPU obtido. Por um lado, é observado que a eficiência diminui quando o tamanho da instância aumenta, por outro, é possível obter uma eficiência maior que 85% com dez processadores. Outro gráfico, desenvolvido por observações empíricas, mostra que pode-se obter resultados com eficiência quando é utilizada uma busca de soluções concorrente (Fig. 4, p. 451). Entretanto, é possível ver que os gráficos, apesar de renderem conclusões, focam numa instância de tamanho 15, e

também não são explicados como foram feitos os estudos empíricos, como por exemplo, se foram observados para todas as instâncias, o mesmo comportamento.

O artigo 3 de Pardalos, Pitsoulis e Resende [86] (Tabela 6.1) faz um relato muito breve dos resultados, somente incluindo tabelas para exposição dos dados, apresentando o mínimo, máximo e valor médio das soluções encontradas, juntamente com o dado de quanto a matriz é esparsa. Contudo, as tabelas não são explicadas. Dados como desvio padrão não são relatados. Os gráficos mostram *speedups* entre a solução encontrada pelo GRASP para PAQ Esparsa sobre GRASP para PAQ Denso. O *speedup* é dado pela razão do tempo gasto para resolver um problema com o algoritmo GRASP-D sobre o algoritmo GRASP-S⁷ (Figuras 5 e 6, p.206, 207). A análise feita mostra que, para instâncias com esparsidade maiores que 0,8, o algoritmo proposto (GRASP-S) obtém resultados até duas vezes mais rápido do que o algoritmo GRASP-D. Entretanto, os dois algoritmos têm um desempenho comparável, quando submetidos a instâncias adequadas.

No artigo 4, de Drezner [28] (Tabela 6.1), as medidas de desempenho utilizadas, mostradas por meio de tabelas, são a melhor solução conhecida, número de vezes que foram encontradas soluções ótimas conhecidas, num total de 20 repetições, porcentagem da solução medida sobre a melhor solução conhecida e tempo de execução dado em minutos. A robustez é mostrada, pois o algoritmo conseguiu ter um desempenho satisfatório para todo o conjunto de instâncias de teste utilizado (Seção 4, p. 329). São apresentadas quatro tabelas, em que a primeira comparou quatro algoritmos, sendo um algoritmo não genético, um algoritmo genético com uma ideia de outro autor e os métodos propostos, chamados de *cohesive* e *scrambled merging* (Tabela 1, p. 326). Foi visto que o algoritmo que não é genético teve o tempo de execução mais longo e que o algoritmo genético de outro autor teve um tempo maior do que os propostos. São relatadas algumas exceções encontradas no tempo médio para algumas instâncias. Com isto, a Tabela 2 mostrou o desempenho destas instâncias, com 100 repetições por instância, e foi concluído que o método *cohesive merging* possui um melhor desempenho que o *scrambled merging*. Visto que o método *cohesive merging* obteve o melhor desempenho, foram modificadas as estratégias de busca de solução, de três maneiras, e comparadas na Tabela 3, p. 328, com os mesmos itens que a Tabela 1. A última tabela (Tabela 4, p. 329) comparou os resultados com os dados de um trabalho publicado de Ahuja, Orlin e Tiwari[1]. Entretanto, os ambientes computacionais são diferentes, e foram citados. Foi visto que, apesar do algoritmo de Ahuja, Orlin e Tiwari ser testado em um computador mais rápido do que de Drezner, este obteve um melhor desempenho, cerca de 20 vezes melhor, tanto na qualidade da solução quanto no tempo computacional. Por fim, não são utilizados gráficos.

⁷O GRASP-D é o algoritmo para matrizes densas, e o GRASP-S é o algoritmo para matrizes esparsas.

Sobre o relato dos resultados, abordados na Seção 3.5, Capítulo 5 e Apêndice A, os seguintes itens são cobertos pelos quatro artigos e apresentados na Tabela 6.6.

E - Relato e Análise dos Resultados					
Itens Recomendados		Artigo 1	Artigo 2	Artigo 3	Artigo 4
1) Medidas de desempenho	a) Qualidade da solução - acurácia com que as soluções são obtidas	×	×	×	×
	b) Esforço computacional - Tempo da melhor solução encontrada		×	×	
	c) Esforço computacional - tempo médio total de execução		×	×	×
	d) Esforço computacional - tempo por fase (se existirem fases)				
	e) Robustez	×	×	×	×
	f) Precisão numérica		×	×	
	g) Quantidade de iterações		×		
	h) Quantidade de chamadas de uma determinada função				
	i) Operações matemáticas				
2) Justificativa das medidas utilizadas		×			
3) Medidas de análise	a) Valor da solução encontrada		×	×	×
	b) Médias		×	×	×
	c) Totais				
	d) Diferenças				
	e) Valor mínimo encontrado			×	
	f) Valor máximo encontrado			×	
	g) Desvio padrão		×		
	h) Erro amostral				
	i) Tamanho da amostra utilizada				
4) Análise estatística	a) Intervalo de confiança				
	b) Nível de confiança				
	c) Margem de erro				
	d) Tamanho da amostra utilizada				
5) Uso de gráficos legíveis			×	×	
6) Uso de tabelas legíveis		×	×	×	×

Tabela 6.6: *Itens cobertos sobre o Relato dos Resultados.*

6.3.6 Conclusões

Por fim, as conclusões devem ser justificadas a partir dos dados apresentados, e se possível, dar direções das pesquisas futuras, possíveis melhoras no algoritmo, identificação das instâncias que não foram resolvidas, problemas e dificuldades encontradas, como por exemplo, casos anômalos (Seção 5.2).

No artigo 1, de Burkard e Rendl [12] (Tabela 6.1), além de concluir que o algoritmo proposto obtém melhores resultados com os que são comparados, pelo comportamento heurístico e pelos parâmetros do método, conclui-se que é possível controlar a eficiência e o tempo computacional controlando, por exemplo, o parâmetro de resfriamento da metaheurística. Também é apresentado ao final do relato uma extensão do algoritmo, que pode ser utilizada para resolver qualquer problema de otimização. O artigo não cita nada sobre trabalhos futuros.

O artigo 2, de Taillard [105] (Tabela 6.1), mostra que foi possível encontrar boas soluções para o PAQ usando a metaheurística Busca Tabu. Uma conclusão obtida é que se são feitas aproximadamente N^2 iterações no método, sendo N o tamanho da instância, e o tamanho da lista tabu tem uma variação de 10% no tamanho da instância, é possível encontrar boas soluções, para instâncias de tamanho maior que 20. O algoritmo trabalha bem para instâncias de tamanho menor do que 30. Para instâncias maiores, uma sugestão é dada: mudar a função de aspiração, adicionando um parâmetro, mas sem mudar a complexidade do algoritmo para que o algoritmo possa encontrar soluções em um tempo computacional viável para instâncias de tamanho maior que 64. Contudo, é concluído que para grandes instâncias é necessário o uso de uma implementação mais sofisticada de Busca Tabu. Esta informação permite que se abram novos horizontes para pesquisas futuras. E também, como contribuição, o gerador de instâncias aleatórias foi desenvolvido de modo a facilitar a experimentação para trabalhos futuros.

O artigo 3 de Pardalos, Pitsoulis e Resende [86] (Tabela 6.1), apesar das conclusões que o algoritmo executa em um tempo aproximadamente 35% mais rápido que o outro algoritmo que não trata a esparsidade das matrizes, e que para matrizes com um esparsidade maior ou igual a 0,8, o algoritmo chega a executar 300% vezes mais rápido, não há nenhuma proposta sobre trabalhos futuros neste relato.

O artigo 4, de Drezner [28] (Tabela 6.1), conclui que sua abordagem para reprodução, que utiliza um algoritmo de busca tabu concêntrico, obteve resultados melhores que a abordagem de outro trabalho, e também sobre um algoritmo que não é baseado em algoritmo genético. Todas as instâncias foram resolvidas com êxito. Drezner cita que para trabalhos futuros, pode-se examinar novas regras de reprodução, e também utilizar variações do algoritmo de busca tabu utilizado, como o proposto por Taillard [105], que é o artigo analisado neste trabalho.

A Tabela 6.7 mostra que os artigos seguiram parcialmente as recomendações para escrita das conclusões de seus experimentos:

F - Conclusões				
Itens Recomendados	Artigo 1	Artigo 2	Artigo 3	Artigo 4
1) Justificar as conclusões a partir dos dados apresentados	×	×	×	×
2) Identificação das instâncias que foram resolvidas com êxito	×	×	×	×
3) Identificação das instâncias que não foram resolvidas				
4) Possíveis melhoras no algoritmo		×		×
5) Direções nas pesquisas futuras		×		×

Tabela 6.7: *Itens cobertos sobre as Conclusões.*

6.4 Conclusões do Capítulo

As revisões bibliográficas deixaram a desejar em relação ao estado da arte, pois apesar de citarem alguns métodos desenvolvidos para o problema, e em sua maioria somente os utilizados para comparação, não mostrando outros trabalhos relacionados. Entretanto, foi visto que autores referenciaram os autores anteriores dos artigos selecionados, como o artigo 2 de Taillard [105] que cita o de Burkard [12]; o artigo 3 de Pardalos, Pitsoulis e Resende não cita nenhum artigo analisado aqui, pois este foi limitado ao estudo de GRASP e citou somente os trabalhos relacionados à metaheurística GRASP; e o artigo 4 cita o de Burkard [12] e de Taillard [105], e não citou o artigo 3 [96].

Em relação ao modelo experimental, foi visto que as principais partes de um relato de experimento, como o modelo experimental, não foram relatadas de maneira lógica nos artigos 1 e 2. Detalhes da condução e procedimentos para execução do experimento computacional, com exceção do artigo 3, não foram citadas. Alguns itens até foram relatados, mas ficaram soltos no texto, pois não houve uma parte no relato dedicada ao experimento. Já os artigos 3 e 4 descreveram a maioria dos itens recomendados, sendo que o 4 tem uma seção dedicada aos experimentos computacionais. O artigo 3 tem uma seção sobre projeto e implementação, ou seja, relatou duas coisas em uma só parte do texto, misturando-as.

Em nenhum trabalho foi utilizado o rigor científico para o planejamento experimental, isto é, foram feitos experimentos totalmente empíricos e não foi utilizado planejamento experimental estatístico. Entretanto, seguiram o modelo experimental básico, que é o método mais utilizado para análise de desempenho e comportamento de algoritmos. Todavia, para execução dos testes vários parâmetros que podem variar, como a quantidade

de testes por instância, não foram relatados nos artigos 1 e 2. Nenhum dos trabalhos citou sobre o uso de testes piloto, que são essenciais para a calibragem inicial dos parâmetros.

Na parte de apresentação dos algoritmos, os artigos apresentaram-os de diferentes maneiras. Como por exemplo, o artigo 2 descreveu partes importantes do algoritmo, mas não apresentou o pseudocódigo do método. Os artigos 1 e 2 apresentaram os algoritmos, já o artigo 4 representou o algoritmo por meio de passos, sem descrever todas as variáveis necessárias e estruturas algorítmicas (laços, condições, etc).

Sobre implementação, a maioria não deu importância a este item, sequer citou a linguagem de programação utilizada, com exceção do artigo 2, que foi o que mostrou todos os detalhes de implementação contidos no *Checklist*. Como todos os métodos desenvolvidos conseguiram alcançar bons resultados, as conclusões deixavam claro isto. Entretanto, falharam sobre mostrar pesquisas futuras.

Para o relato e análise dos resultados, o fator principal é a exposição dos resultados para que possam ser comparados no futuro. O artigo 1 conclui que o algoritmo proposto dá resultados melhores, mas como não é explicado como foram feitos os testes, fica complicado reproduzir o experimento, tornando os resultados duvidosos. Nos artigos 2 e 3 ocorre a mesma coisa. O único que explica os resultados é o artigo 4, calculando porcentagem média das soluções encontradas, explicando qual algoritmo obteve melhor resultado, com quais instâncias foram obtidas melhores desempenho nos testes e que tiveram um desempenho menor. De um modo geral, as tabelas poderiam ser mais completas e melhor explicadas no que diz respeito às características das instâncias e da qualidade dos resultados obtidos, incluindo cálculos estatísticos sobre os resultados.

Como os algoritmos propostos obtiveram bons resultados, segundo os testes realizados, as justificativas vieram das tabelas com os dados de comparação expostos. Os artigos 1 e 3 falharam em apresentar trabalhos futuros, e também não concluíram que a pesquisa sobre o algoritmo proposto estava concluída, ou seja, ficou em aberto a questão sobre pesquisas futuras.

Considerações Finais

Este trabalho apresentou um estudo sobre condução e relato de experimentos computacionais, especificamente em relação à métodos heurísticos. Foi visto que testes computacionais realizados com algoritmos são feitos por algumas razões, dentre elas demonstrar: a corretude do modelo formal do problema resolvido pelo algoritmo, a qualidade das soluções, a velocidade de computação ou a robustez do algoritmo. Enfim, compreender o comportamento de algoritmos para então aplicá-los a problemas reais.

Dado que os resultados de estudos experimentais na computação podem influenciar aplicações do mundo real, é de suma importância saber prepará-los, realizá-los e relatá-los com o máximo de cuidado. Apesar disso, pode-se identificar em diversos trabalhos muitas falhas no relato de experimentos que poderiam ser evitadas com o uso de boas práticas em sua condução. Além disso, é sempre desejável, para melhor entendimento dos experimentos, que o nível de qualidade dos artigos e relatórios seja o maior possível.

As principais recomendações sobre condução de experimentos são, de um modo geral, bastante conhecidas [6, 19, 57, 67, 76, 95], pois os estudos nessa área datam desde 1979, como o trabalho pioneiro de Crowder, Dembo e Mulvey [19]. Contudo, tais estudos não definem métricas nem diretrizes para pesquisas deste tipo, ou seja, o que existe são apenas recomendações de boas práticas.

Desta forma, a dissertação descreve, nos Capítulos 3 a 5, os principais aspectos que devem ser considerados na condução e relato de experimentos computacionais, e um estudo exemplo no Capítulo 6. Com isso, as principais contribuições do estudo aqui apresentado são:

- Extensa investigação sobre condução de experimentos computacionais;
- Identificação de um conjunto de recomendações para melhorar a condução de um experimento computacional;
- Organização das várias recomendações encontradas, que estavam fragmentadas na literatura;
- Elaboração de um *checklist* (Tabela 5.2), representando de forma sumariada todos os itens vistos nesta investigação;

- Estudo exemplo de relatos consolidados na literatura para mostrar a aplicação das recomendações encontradas e citadas neste trabalho.

O trabalho de Crowder, Dembo e Mulvey [19] apresenta um *checklist* de pontos a considerar quando avaliar ou relatar um experimento computacional, colocando itens que são fortemente recomendados e outros que são opcionais (Ver Tabela B.1). Uma primeira análise dos artigos considerados no Capítulo 6 (Vide Tabela 6.1), foi feita utilizando como guia esta tabela. Entretanto, observou-se que tais itens não são muito pontuais, abrangendo em um único item várias recomendações, o que dificultou a validação de determinados itens. Então, foram considerados todos os itens vistos nesta pesquisa e um novo guia foi produzido, que é o *checklist* proposto neste trabalho, contido na Tabela 5.2, que divide os itens de forma mais clara e objetiva. O estudo exemplo foi refeito com este *checklist* e resultado da análise foi mais conclusiva a respeito dos itens cobertos por cada artigo.

Uma conclusão obtida com a análise destes artigos é que determinados autores dão importância a alguns itens recomendados e outros não. O *checklist* proposto permite uma grande flexibilidade para análise de artigos. Uma sugestão para aplicação deste é atribuir pesos aos itens de acordo com a análise requerida. Por exemplo, se o modelo experimental é considerado mais importante que dados sobre implementação, dá-se um peso maior aos itens contidos no modelo experimental. Perceba que no estudo do Capítulo 6, não foram atribuídos pesos aos itens recomendados, somente foi visto se os relatos seguiram ou não determinada recomendação, que pode ser outra opção de análise, isto é, simplesmente verificar se o item foi coberto ou não.

O *checklist* proposto pode ter várias aplicações, dentre elas um guia para elaboração de relato de experimentos, pode ser utilizado por revisores para avaliação de artigos em conferências, e também para outras aplicações que requerem relatos, bem como condução de experimentos, não somente em Otimização. Se for utilizado para avaliação de artigos, esta tende a ser mais justa e não tão superficial, devido aos critérios estabelecidos que o trabalho deverá atender. Sendo assim, o *checklist* pode ser adaptado para cada caso.

Para trabalhos futuros, algumas propostas foram levantadas:

- Expandir o estudo para algoritmos exatos;
- Expandir o estudo para algoritmos paralelos e distribuídos, que necessitam mais medidas de análise;
- Detalhar recomendações dos Capítulos 3 a 5, como por exemplo, a análise estatística;
- Comparar trabalhos que contém modelo experimental básico e modelo experimental estatístico, e analisar quais as contribuições que a análise estatística rigorosa pode oferecer;

- Utilizando o *checklist*, escolher um problema e um subconjunto de artigos deste, refazer os testes ou reimplementar os algoritmos, e comparar qualitativamente os resultados dos artigos com os novos testes;
- Verificar a evolução cronológica do relato de artigos. Para isso é necessário escolher um problema, selecionar uma grande quantidade de artigos, analisá-los em relação ao *checklist* e fazer análise estatística sobre os dados. Com isso, seria possível saber se houve ou não uma evolução no relato de experimentos computacionais para o problema em questão.

Referências Bibliográficas

- [1] AHUJA, R. K.; ORLIN, J. B.; TIWARI, A. **A descent genetic algorithm for the quadratic assignment problem.** *Computers and Operational Research*, 27:917–934, 2000.
- [2] AHUJA, R. K.; MAGNANTI, T. L.; ORLIN, J. B. **Network Flows: theory, algorithms, and applications.** Prentice-Hall, 1993.
- [3] AHUJA, R. K.; ORLIN, J. B. **Use of representative operation counts in computational testings of algorithms.** Working papers 3459-92, Massachusetts Institute of Technology (MIT), Sloan School of Management, 1992.
- [4] ALBA, E.; BLUM, C.; ROLI, A. **An Introduction to Metaheuristic Techniques.** In: *Parallel Metaheuristics: A New Class of Algorithms*, Wiley Series on Parallel and Distributed Computing, chapter 1. Wiley-Interscience, 2005.
- [5] ALVES, J. S. **Times Assíncronos Dinâmicos.** Master's thesis, Universidade Federal de Goiás, Goiânia, 2009.
- [6] BARR, R. S.; GOLDEN, B. L.; KELLY, J. P.; RESENDE, M. G. C.; STEWART, W. R. **Designing and Reporting on Computational Experiments with Heuristic Methods.** *Journal of Heuristics*, 1:9–32, 1995.
- [7] BEASLEY, J. E. **OR-Library: Distributing Test Problems by Electronic Mail.** *Journal of the Operational Research Society*, 41(11):1069–1072, Novembro 1990.
- [8] BISQUERRA, R.; SARRIERA, J. C.; MARTÍNEZ, F. **Introdução à Estatística: Enfoque Informático com o Pacote Estatístico SSPS.** Artmed, Porto Alegre, 2004.
- [9] BOX, G. E. P.; HUNTER, W. G.; HUNTER, J. S. **Statistics for Experimenters : an Introduction to Design, Data Analysis and Model Building.** John Wiley & Sons, Inc., 1978.

- [10] BRUNS, R. E.; DE BARROS NETO, B.; SCARMINIO, I. S. **Como Fazer Experimentos: Pesquisa e Desenvolvimento na Ciência e na Indústria**. Editora da Unicamp, 2 edition, 2003.
- [11] BURKARD, R. E.; KARISCH, S. E.; RENDL, F. **QAPLIB - A Quadratic Assignment Problem Library**. <http://www.opt.math.tugraz.at/qaplib/>, último acesso em Fevereiro de 2011, 1996.
- [12] BURKARD, R. E.; RENDL, F. **A thermodynamically motivated simulation procedure for combinatorial optimization problems**. *European Journal of Operational Research*, 17(2):169–174, August 1984.
- [13] CALADO, V.; MONTGOMERY, D. C. **Planejamento de Experimentos usando o Statistica**. E-Papers Serviços Editoriais, Rio de Janeiro, 2003.
- [14] CLEVELAND, W. S. **Visualizing data**. AT & Bell Laboratories, 1993.
- [15] COFFIN, M.; SALTZMAN, M. J. **Statistical analysis of computational tests of algorithms and heuristics**. *INFORMS Journal on Computing*, 12(1):24–44, 2000.
- [16] COMMANDER, C. W. **A survey of the quadratic assignment problem, with applications**. *Morehead Electronic Journal of Applicable Mathematics*, 4(1):1–15, 2005.
- [17] CORDON, O.; HERRERA, F.; STÜTZLE, T. **A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends**. *Mathware & Soft Computing*, 9(2–3):141–175, 2002.
- [18] CORMEN, T.; LEISERSON, C.; RIVEST, R.; STEIN, C. **Introduction to Algorithms**. MIT Press, 2001.
- [19] CROWDER, H.; DEMBO, R. S.; MULVEY, J. M. **On reporting computational experiments with mathematical software**. *ACM Transactions on Mathematical Software*, 5(2):193–203, 1979.
- [20] CROWDER, H. P.; DEMBO, R. S.; MULVEY, J. M. **Reporting computational experiments in mathematical programming**. *Mathematical Programming*, 15(1):316–329, 1978.
- [21] DE ALMEIDA CALEGARE, A. J. **Introdução ao Delineamento de Experimentos**. Editora Blucher, 2 edition, 2009.
- [22] DE WERRA, D.; HERTZ, A. **Tabu Search Techniques: A Tutorial and an Application to Neural Networks**. *OR Spectrum*, 11(3):131–141, 1989.

- [23] DORIGO, M.; MANIEZZO, V.; COLORNI, A. **Ant System: Optimization by a Colony of Cooperating Agents.** *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, Fevereiro 1996.
- [24] DORIGO, M.; BLUM, C. **Ant Colony Optimization Theory: a Survey.** *Theoretical Computer Science*, 344(2-3):243–278, 2005.
- [25] DORIGO, M.; SOCHA, K. **An Introduction to Ant Colony Optimization.** Technical Report TR/IRIDIA/2006-010, Université Libre de Bruxelles, Abril 2006.
- [26] DORIGO, M.; STÜTZLE, T. **The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances.** In: *Handbook of Metaheuristics*, chapter 9, p. 250–285. International Series in Operations Research & Management Science, 2003.
- [27] DOWSLAND, K. A. **Simulated annealing.** In: *Modern Heuristic Techniques for Combinatorial Problems*, chapter 2, p. 20–69. John Wiley & Sons, Inc., New York, NY, USA, 1993.
- [28] DREZNER, Z. **A New Genetic Algorithm for the Quadratic Assignment Problem.** *INFORMS JOURNAL ON COMPUTING*, 15(3):320–330, 2003.
- [29] DREZNER, Z.; HAHN, P. M.; TAILLARD, É. D. **Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods.** *Annals OR*, 139(1):65–94, 2005.
- [30] DYER, M. E.; FRIEZE, A. M. **A Simple Heuristic for the p-Centre Problem.** *Operations Research Letters*, 3(6):285–288, 1985.
- [31] FEO, T. A.; RESENDE, M. G. **Greedy Randomized Adaptive Search Procedures.** *Journal of Global Optimization*, 6:109–133, 1995.
- [32] FISHER, R. A.; TIPPETT, L. H. C. **Limiting forms of the frequency distribution of the largest or smallest member of a sample.** *Mathematical Proceedings of the Cambridge Philosophical Society*, 24:180–190, 1928.
- [33] FRIGON, N. L.; MATHEWS, D. **Practical Guide to Experimental Design.** John Wiley & Sons, Inc., 1997.
- [34] GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability : a Guide to the Theory of NP - Completeness.** W. H. Freeman, New York, 1979.
- [35] GENDREAU, M.; POTVIN, J. **An introduction to tabu search.** In: Glover, F., editor, *Handbook of Metaheuristics*, chapter 2, p. 37–54. Kluwer Academic Publishers, 2003.

- [36] GHASHGHAI, E.; RARDIN, R. L. **Using a hybrid of exact and genetic algorithms to design survivable networks.** *Computers & Operations Research*, 29(1):53–66, 2002.
- [37] GLOVER, F. **Future Paths for Integer Programming and Links to Artificial Intelligence.** *Computers and Operations Research*, 13(5):553–549, 1986.
- [38] GLOVER, F. **Tabu Search -Part I.** *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [39] GLOVER, F. **Tabu Search -Part II.** *ORSA Journal on Computing*, 2(1):4–32, 1989.
- [40] GLOVER, F. **Tabu Search and Adaptative Memory Programing - Advances, Applications and Challenges.** In: Barr, R.; Helgason, R.; Kennington, J., editors, *Interfaces in Computer Science and Operations Research*, p. 1–75. Kluwer Academic Publishers, 1996.
- [41] GOLDBERG, D. E. **Genetics Algorithms in Search, Optimization, and Machine Learning.** Addison-Wesley, Berkeley, 1989.
- [42] GOLDEN, B. L. **A Statistical Approach to the TSP.** *Networks*, 7:209–225, 1977.
- [43] GREENBERG, H. J. **Computational Testing: Why, How and How Much.** *INFORMS Journal on Computing*, 2(1):94–97, 1990.
- [44] HANSEN, P. **The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming.** Capri, 1986.
- [45] HANSEN, P.; MLADENOVIC, N. **A Tutorial on Variable Neighborhood Search.** Technical Report G-2003-46, Les Cahiers du GERAD, Julho 2003.
- [46] HANSEN, P.; MLADENOVIC, N. **Variable Neighborhood Search.** In: Glover, F. W.; Kochenberger, G. A., editors, *Handbook of Metaheuristics*, chapter 7, p. 145–184. Kluwer Academic Publishers, 2003.
- [47] HELD, M.; KARP, R. M. **The Traveling-Salesman Problem and Minimum Spanning Trees.** *Operations Research*, 18(6):1138–1162, 1970.
- [48] HELD, M.; KARP, R. M. **The Traveling-Salesman Problem and Minimum Spanning Trees: Part II.** *Mathematical Programming*, 1(1):6–25, Dezembro 1971.
- [49] HERTZ, A.; DE WERRA, D. **The Tabu Search Metaheuristic: How We Used It.** *Annal of Mathematics and Artificial Intelligence*, 1:111–121, 1990.

- [50] HERTZ, A.; TAILLARD, E.; DE WERRA, D. **A Tutorial on Tabu Search**. Technical report, EPFL, Département de Mathématiques, MA-Ecublens, CH-1015, Lausanne, 1992.
- [51] HOCHBAUM, D. S.; SHMOYS, D. B. **A Best Possible Heuristic for the k-Center Problem**. *Mathematics of Operations Research*, 10(2):180–184, 1985.
- [52] HOLLAND, J. **Adaptation in Natural and Artificial Systems**. University of Michigan Press, 1975.
- [53] HOOKER, J. **Needed: An Empirical Science Of Algorithms**. *Operations Research*, 42:201–212, 1994.
- [54] HOOKER, J. **Testing Heuristics: We Have It All Wrong**. *Journal of Heuristics*, 1:33–42, 1995.
- [55] HOPFIELD, J. J.; TANK, D. W. **Neural Computation of Decisions in Optimization Problems**. *Biological Cybernetics*, 52(3):141–152, Julho 1985.
- [56] JOHNSON, D.; PAPADIMITRIOU, C. I. L. E.; LENSTRA, J.; KAN, A. R.; SHMOYS, D. **Performance Guarantees for Heuristics**, p. 145–180. John Wiley e Sons Ltd., Chichester(UK), 1985.
- [57] JOHNSON, D. S. **A Theoretician's Guide to the Experimental Analysis of Algorithms**. In: *Dagstuhl Seminar on Experimental Algorithmics*, Setembro 2001.
- [58] KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. **Optimization by Simulated Annealing**. *Science*, 220(4598):671–680, 1983.
- [59] KOOPMANS, T. C.; BECKMANN, M. J. **Assignment problems and the location of economic activities**. Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University, 1957.
- [60] LILJA, D. J. **Measuring Computer Performance, A Practitioner's Guide**. Cambridge University Press, 1 edition, 2004.
- [61] LIN, S.; KERNIGHAN, B. W. **An Effective Heuristic Algorithm for the Traveling-Salesman Problem**. *Operations Research*, 21(2):498–516, 1973.
- [62] LOIOLA, E. M.; DE ABREU, N. M. M.; BOAVENTURA-NETTO, P. O.; HAHN, P.; QUERIDO, T. **A survey for the quadratic assignment problem**. *European Journal of Operational Research*, 176(2):657–690, January 2007.

- [63] LOS, M.; LARDINOIS, C. **Combinatorial programming, statistical optimization and the optimal transportation network problem.** *Transportation Research Part B: Methodological*, 16(2):89–124, 1982.
- [64] LUGER, G. F. **Inteligência Artificial: Estruturas e Estratégias para a Solução de Problemas Complexos.** Bookman, Porto Alegre, 2004.
- [65] **MathWorks - Accelerating the pace of engineering and science.** <http://www.mathworks.com/products/statistics/>, último acesso em Fevereiro de 2011, 1994.
- [66] MCGEOCH, C. C. **Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups.** *ACM Computing Surveys*, 24(2):195–212, 1992.
- [67] MCGEOCH, C. C. **Toward an Experimental Method for Algorithm Simulation.** *INFORMS Journal on Computing*, 8(1):1–15, 1996.
- [68] MCGEOCH, C. C.; MORET, B. M. E. **How to Present a Paper on Experimental Work with Algorithms.** *SIGACT News*, 30(4):85–90, 1999.
- [69] MENDES, J. T.; E SILVA, L. H. M.; SPERANDIO, D. **Cálculo Numérico: Características Matemáticas e Computacionais dos Métodos Numéricos.** Prentice-Hall, 2003.
- [70] METROPOLIS, N.; ROSENBLUTH, A.; ROSENBLUTH, M.; TELLER, A.; TELLER, E. **Equation of State Calculation by Fast Computing Machines.** *The Journal of Chemical Physics*, 21(6):1807–1092, 1953.
- [71] MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs.** Springer-Verlag New York, Inc., New York, NY, USA, 3 extended edition, 1996.
- [72] MICHIELS, W.; AARTS, E.; KORST, J. **Theoretical Aspects of Local Search.** Springer, Verlag New York Inc, 2007.
- [73] **Minitab 16 - Statistical Software.** <http://www.liderssoftwares.com.br/minitab/>, último acesso em Fevereiro de 2011, 2010.
- [74] MLADENOVIC, N.; HANSEN, P. **Variable Neighborhood Search.** *Computers & Operations Research*, 24(11):1097–1100, 1997.
- [75] MONTGOMERY, D. C. **Design and Analysis of Experiments.** John Wiley & Sons, Inc., 7 edition, 2009.

- [76] MORET, B. M. E. **Towards a discipline of experimental algorithmics.** In: Goldwasser, M. H.; Johnson, D. S.; McGeoch, C. C., editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59 de **DIMACS Monographs**, p. 197–213. AMS Press, 2002.
- [77] MOTWANI, R. **Lecture Notes on Approximation Algorithms: Volume I.** Technical report, Stanford University, Stanford, CA, USA, 1993.
- [78] NEHI, H. M.; GELAREH, S. **A survey of meta-heuristic solution methods for the quadratic assignment problem.** *Applied Mathematical Sciences*, 1(46):2293–2312, 2007.
- [79] **Netlib Repository at UTK and ORNL.** <http://www.netlib.org/>, último acesso em Janeiro de 2010, 2010.
- [80] **Octave.** <http://www.gnu.org/software/octave/>, último acesso em Fevereiro de 2011, 1998.
- [81] OLIVEIRA, P. H. F. C. **Amostragem Básica: Uma Aplicação em Auditoria.** Editora Ciência Moderna, 2004.
- [82] **Welcome to OR-Library.** <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>, último acesso em Março de 2010, 2010.
- [83] OSMAN, I.; LAPORTE, G. **Meta-heuristics: A Bibliography.** *Annals of Operations Research*, 63(5):511–623, 1996.
- [84] OVACIK, I. M.; RAJAGOPALAN, S.; UZSOY, R. **Integrating interval estimates of global optima and local search methods for combinatorial optimization problems.** *Journal of Heuristics*, 6(4):481–500, 2000.
- [85] PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization: Algorithms and Complexity.** Prentice-Hall, 1982.
- [86] PARDALOS, P. M.; PITSOULIS, L. S.; RESENDE, M. G. C. **Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using grasp.** *ACM Trans. Math. Softw.*, 23(2):196–208, June 1997.
- [87] PARDALOS, P. M.; RENDL, F.; WOLKOWICZ, H. **The quadratic assignment problem: A survey and recent developments.** In: *In Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, p. 1–42. American Mathematical Society, 1994.

- [88] PEIXOTO, H. P. **Metodologia de Especificação de Times Assíncronos para Problemas de Otimização Combinatória**. Master's thesis, UNICAMP, Março 1995.
- [89] PEIXOTO, H. P.; SOUZA, P. S. **Uma Metodologia de Especificação de Times Assíncronos**. Technical Report DCC-94-12, Universidade Estadual de Campinas, Novembro 1994.
- [90] PONTES, F. J.; FERREIRA, J. R.; DE PAIVA, A. P.; BALESTRASSI, P. P. **Aplicação de um Arranjo Fatorial Completo ao Projeto dos Parâmetros de uma Rede Neural Artificial RBF para Predição da Rugosidade em Processo de Torneamento**. *XXVII Encontro Nacional de Engenharia de Produção*, p. 1–10, Outubro 2007.
- [91] POYLA, G. **A arte de Resolver Problemas: um Novo Aspecto do Método Matemático**. Interciência, Rio de Janeiro, 2006.
- [92] PRAIS, M.; RIBEIRO, C. C. **Variação de Parâmetros em Procedimentos GRASP**. *Investigación Operativa*, 9:1–20, 2000.
- [93] PUCHINGER, J.; RAIDL, G. R. **Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification**. In: *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, p. 41–53, 2005.
- [94] RARDIN, R. L.; TOWEY, C. A.; PILCHER, M. G. **Analysis of a Random Cut Test Instance Generator for the TSP**, p. 387–405. World Scientific Publishing, 1993.
- [95] RARDIN, R. L.; UZSOY, R. **Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial**. *Journal of Heuristics*, 7(3):261–304, 2001.
- [96] RESENDE, M. G. C.; PARDALOS, P. M.; LI, Y. **Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using grasp**. *ACM Trans. Math. Softw.*, 22(1):104–118, March 1996.
- [97] ROSA, M.; OREY, D. C. **De Pappus a Polya: Da Heurística Grega a Modelagem Matemática**. *Plures. Humanidades*, p. 1–17, 2009.
- [98] ROSSETI, I. C. M. **Estratégias Sequenciais e Paralelas de GRASP com Reconexão por Caminhos para o Problema de Síntese de Redes a 2-Caminhos**. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2003.
- [99] ROTHFARB, B.; FRANK, H.; ROSENBAUM, D. M.; STEIGLITZ, K.; KLEITMAN, D. J. **Optimal Design of Offshore Natural-Gas Pipeline Systems**. *Operations Research*, 18(6):992–1020, 1970.

- [100] **Scilab - The Free Software for Numerical Computation.** <http://www.scilab.org/products/scilab/download>, último acesso em Fevereiro de 2011, 1989.
- [101] SHMYGELSKA, A.; AGUIRRE-HERNÁNDEZ, R.; HOOS, H. H. **An ant colony optimization algorithm for the 2D HP protein folding problem.** In: Dorigo, M.; Caro, G. D.; Sampels, M., editors, *Ant Algorithms, Third International Workshop, ANTS 2002, Brussels, Belgium, September 12-14, 2002, Proceedings*, volume 2463 de **Lecture Notes in Computer Science**, p. 40–53. Springer, 2002.
- [102] SHOKRANIAN, S. **Tópicos em Métodos Computacionais.** Editora Ciência Moderna, 2009.
- [103] SOUZA, M. J. F. **Inteligência Computacional para Otimização.** Technical report, Universidade Federal de Ouro Preto, 2009.
- [104] SOUZA, P. S.; TALUKDAR, S. N. **Asynchronous Organizations for Multi-algorithm Problems.** In: *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, p. 286–293, New York, NY, USA, 1993. ACM.
- [105] TAILLARD, E. **Robust taboo search for the quadratic assignment problem.** *Parallel Computing*, 17(4-5):443–455, 1991.
- [106] TALUKDAR, S. N.; SOUZA, P. S. **Scale Efficient Organizations.** In: *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, p. 1458–1463 vol.2, 18-21 1992.
- [107] TRIOLA, M. F. **Introdução à Estatística.** LTC - Livros Técnicos e Científicos Editora S.A., 7 edition, 1999.
- [108] **TSPLIB.** <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/>, último acesso em Janeiro de 2010, 2010.
- [109] VIANA, G. V. R. **Meta-heurísticas e Programação Paralela em Otimização Combinatória.** Editora EUFC, 1998.
- [110] VOSS, S. **Meta-heuristics: The State of the Art.** In: *ECAI '00: Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers*, p. 1–23, London, UK, 2001. Springer-Verlag.
- [111] ZANAKIS, S. H.; EVANS, J. R.; VAZACOPOULOS, A. A. **Heuristic Methods and Applications: A Categorized Survey.** *European Journal of Operational Research*, 43(1):88–110, Novembro 1989.

Conceitos Básicos de Estatística

Este apêndice apresenta alguns conceitos básicos de Estatística. Como este trabalho fala sobre condução de experimentos, a Seção A.1 define o que é Planejamento Experimental, princípios básicos e aplicações. A Seção A.2 descreve o que é população, amostra, fator, entre outras definições utilizadas no planejamento de experimentos. A Seção A.3 mostra os cálculos estatísticos básicos utilizados na análise de experimentos, tais como média, variância e desvio padrão, etc. Por fim, a Seção A.4 dá uma breve explicação sobre os métodos estatísticos Análise de Variância de um Fator, Fatorial Completo e Quadrado Latino. Este apêndice foi baseado em [8, 13, 21, 33, 54, 75, 81, 95, 107].

A.1 Planejamento Experimental

Em qualquer pesquisa, em que são feitos experimentos, é gerado um conjunto de dados. Para terem um valor significativo e levar a conclusões corretas, são utilizadas técnicas estatísticas. Todo este processo é definido como Planejamento Experimental.

Calegare [21] e Montgomery [75] definem **Planejamento Experimental** (ou Delineamento de Experimentos) como o plano formal para conduzir o experimento. Inclui a escolha de fatores, níveis, tratamentos e número de réplicas. Pode ser definido como uma estratégia a ser utilizada num estudo empírico. De acordo com Calado e Montgomery [13], os princípios básicos de um planejamento de experimentos são: **replicação, aleatoriedade e blocagem**.

- **Replicação:** Consiste na repetição ou duplicação de um experimento para que os resultados possam ser confirmados ou verificados. Fazer réplicas permite a obtenção do erro experimental, pois a estimativa deste erro serve para verificar se as diferenças observadas nos dados são estatisticamente diferentes. A replicação também permite a obtenção de uma estimativa mais precisa de um determinado fator, caso a média de uma amostra seja usada para estimar o efeito de um fator no experimento.

- **Aleatoriedade:** Diz respeito a como os dados são coletados e como são realizados os experimentos. Os dados devem ser coletados de um modo adequado e com a mesma probabilidade de serem selecionados. Caso isso não seja feito, os dados podem ser tornar inúteis. Portanto, os experimentos, com suas réplicas, devem ser realizados de maneira aleatória, de modo a garantir a distribuição imparcial de todos os fatores.
- **Blocagem (Planejamento em Blocos):** É uma técnica utilizada quando determinados fatores têm um efeito considerável no experimento. Para isso, usam-se blocos para agrupar sujeitos semelhantes, mas que podem afetar o resultado do experimento de uma maneira diferente. É muito usada industrialmente, com o objetivo de aumentar a precisão de um experimento. A blocagem também é usada quando uma determinada medida experimental é feita por diferentes pessoas, levando a uma possível não homogeneidade nos dados.

Antes de começar a realizar os experimentos, é muito útil definir os objetivos e os critérios, que devem estar bem claros e justificados, como por exemplo:

- As variáveis envolvidas nos experimentos;
- A faixa de variação das variáveis selecionadas;
- Os níveis escolhidos para essas variáveis. No caso de muitos fatores, é melhor escolher inicialmente dois níveis;
- A variável resposta;
- O planejamento experimental, pois nesta etapa, deve-se considerar o tamanho da amostra, a seleção de uma ordem para a realização dos experimentos e também se há vantagem em fazer a blocagem dos experimentos. Por fim, escolher quais os métodos de análise dos resultados dos experimentos serão utilizados. Os métodos estatísticos são usados para guiar uma tomada objetiva de decisão.

Todo planejamento experimental começa com uma série de experimentos. Geralmente, a primeira rodada de experimentos é chamada de **experimento exploratório** ou **piloto**. Seu objetivo é determinar quais são as variáveis críticas, ou seja, quais são mais importantes. Os experimentos seguintes são usados para definir os níveis das variáveis críticas identificadas anteriormente, que resultam em um melhor desempenho do processo.

Em resumo, a meta principal da realização de um experimento deste tipo é obter um modelo matemático apropriado para descrever um certo fenômeno, utilizando o mínimo possível de experimentos. O planejamento experimental permite eficiência e economia no processo experimental e o uso de métodos estatísticos na análise dos dados obtidos resulta em objetividade científica nas conclusões.

Algumas **aplicações** do planejamento de experimentos são:

- Avaliação e comparação de configurações básicas de um projeto;
- Avaliação de diferentes materiais;
- Seleção de parâmetros de um projeto;
- Determinação de parâmetros de um projeto que melhorem o desempenho de produtos;
- Obtenção de produtos que sejam mais fáceis de fabricar, que sejam projetados, desenvolvidos e produzidos em menos tempo, que tenham melhor desempenho e confiabilidade que os produzidos pelos concorrentes.

Dentre os modelos experimentais existentes, podem-se citar: ANOVA, Fatorial, Quadrado Latino, entre outros. A seleção de um modelo experimental depende da quantidade de tratamentos a serem avaliados e do custo associado com as execuções do experimento.

Para melhor compreensão do planejamento experimental, os conceitos básicos são descritos a seguir, a Seção A.2, os cálculos estatísticos básicos, Seção A.3, e por fim os métodos estatísticos para análise Análise de Variância, Fatorial e Quadrado Latino, Seção A.4.

A.2 Princípios básicos de um planejamento de experimentos

População é o conjunto de todos os indivíduos que possuem no mínimo uma característica em comum entre eles e nos quais se deseja estudar um fenômeno. A população pode ser finita ou infinita (representada por N). **Amostra** é um subconjunto da população, selecionado por algum método de amostragem, coletados para o estudo do fenômeno (representada por n). Um **indivíduo** é um elemento que compõe a população e também a amostra. Um **parâmetro** é o valor da população, que é constante e geralmente estimado.

Um **fator** (ou variável) é uma característica que pode possuir diferentes valores, isto é, qualquer variável controlável em um experimento que influencia a saída ou resultado deste, ou seja, é uma variável cujo efeito é de interesse da investigação. Os **níveis de um fator** são os valores que estes assumem no experimento. Os fatores podem ser **qualitativos** ou **quantitativos**. Um **fator qualitativo** possui diferentes categorias para cada modalidade da variável, ou seja, são características que não podem ser quantificadas, como por exemplo, um fator que contém o valor verdadeiro ou falso. Já **fator quantitativo** é um fator que pode ser associado a uma escala numérica, tais como temperatura, pressão ou vento. Os fatores quantitativos podem ser divididos em **discretos**, podem assumir determinados valores, como quantidade de cidades ou quantidade de alunos; ou podem

ser **contínuos**, que podem assumir qualquer valor intermediário dentro de uma faixa de valores, tais como peso, rendimento acadêmico, etc.

Os fatores podem ser classificados em **fatores independentes** (também chamados de variável independente ou variável preditora), que são a suposta causa de uma modificação em uma relação de causa e efeito, e são manipulados e controlados pelo pesquisador; **fatores dependentes** (também chamados de variável dependente ou **variável resposta**), é consequência do fator independente; **fatores intervenientes**, que são fatores alheios ao experimento e podem influenciar nos resultados.

Um **tratamento** é um nível único assinalado para um fator durante um experimento. Podem ser divididos em: **aleatórios**, em que o pesquisador não tem possibilidade de manipulá-los e somente pode constatar os valores observados; **controlados**, aqueles cujos valores o pesquisador determina ou atribui a cada um dos indivíduos; **externos**, que são valores que interferem no fenômeno pesquisado, sem serem explicitamente controlados.

Um **ensaio** é cada realização do experimento em uma determinada combinação de tratamentos. O experimento é constituído de todos os ensaios realizados nas diversas combinações de tratamentos, com várias réplicas. Uma **réplica** é uma repetição de um experimento, executada nas mesmas condições experimentais utilizadas.

Geralmente, quando os parâmetros de determinada população não são conhecidos, é retirada uma amostra, e a partir dela são levantadas informações para aceitar ou não o valor hipotético inicial. Um experimento inicia com duas hipóteses, chamadas de **hipóteses iniciais**.

- **Hipótese Nula (H_0):** Hipótese que está sendo testada. Admite-se que a diferença entre o valor obtido na amostra (estimador) e o parâmetro da população não é significativa.
- **Hipótese Alternativa (H_1):** Qualquer hipótese diferente da hipótese nula, ou seja, há diferença entre o estimador e o parâmetro da população.

A amostragem pode ser estatística ou não-estatística. Na primeira, as conclusões valem para Amostra e População. Já na segunda as conclusões só valem para a amostra. As amostras podem ser tipificadas de diversas maneiras: quanto ao seu dimensionamento, quanto ao número de estágios, quanto à seleção das unidades, quanto à organização da população, etc.

Seleção da Amostra

Como a amostra deve ser um retrato da população, ela necessita de critérios para sua criação. A seleção amostral obedece a lei da equiprobabilidade, que diz que todos os

elementos têm a mesma probabilidade $P = \frac{1}{N}$ de serem selecionados. Para tanto, usa-se um procedimento aleatório de seleção.

A amostragem com **seleção aleatória simples** consiste na seleção de n unidades amostrais de tal forma que cada amostra tenha a mesma chance de ser escolhida.

A amostragem com **seleção aleatória sistemática** consiste na seleção por saltos, com o objetivo de varrer toda a população. Para utilizá-la deve-se ter um cadastro da população e esta deve estar ordenada por algum critério, seja quantitativa, cronológica ou qualitativa.

Estimador

Há diferença entre medida amostral e medida populacional. Tudo aquilo que se refere à amostra tem o nome de **estimador**, seu respectivo valor é chamado de **estimativa**, representado por θ .

Por exemplo, a média de uma amostra não é chamada de parâmetro, e sim de estimativa de média populacional, visto que esta não é a sumarização dos dados de um universo.

A **estimativa de parâmetros** relativos a variáveis é feita para que, a partir dos valores obtidos em uma amostra probabilística, seja possível inferir para a população, construindo-se um intervalo de confiança, por meio de um plano amostral e sua execução.

A.3 Cálculos Estatísticos Básicos

A seguir serão detalhados como são feitos os principais cálculos estatísticos, tais como média, desvio padrão, variância, covariância e outros, que são utilizados na comparação de desempenho de algoritmos. Esta seção foi baseada em [9, 10, 69, 81, 102].

Primeiramente, é importante saber o conceito de **erro experimental**. Este erro diz respeito às unidades experimentais, e a existência de um erro está relacionada ao fato de que na natureza não existem unidades experimentais perfeitamente iguais. O erro experimental é também um reflexo de falhas humanas na tentativa de reproduzir com exatidão determinado procedimento. Qualquer medida está sempre afetada por erros. Se os erros forem insignificantes, tudo bem. Entretanto, se não forem, há o risco de fazer inferências incorretas a partir dos resultados experimentais e, possivelmente, chegar a um resultado ou resposta falsa para o experimento. Para que isto não ocorra, deve-se saber como trabalhar com estes erros, tanto na análise do experimento quanto no próprio planejamento, pois não existe análise que possa salvar um experimento mal planejado.

Existem três tipos de erros: **erros grosseiros**, **erros sistemáticos**, **erros aleatórios**. Um exemplo de erro grosseiro em um experimento computacional seria deixar uma

condição de um laço iterar infinitamente. Com isto, o programa nunca terá uma saída, visto que está em *loop* infinito. A estatística não se ocupa desses erros, pois nem existe ciência capaz de tratá-los.

No erro sistemático, o erro afeta o resultado sempre na mesma direção, seja para mais, ou para menos. Um exemplo deste erro seria a má inicialização de variáveis. Cada valor da variável exercerá individualmente sua influência no resultado final, fazendo-o tender para uma certa direção.

Já nos erros aleatórios, pode ocorrer que, a cada execução do algoritmo, haja um resultado completamente diferente dos anteriores, ou que os valores obtidos flutuem, mas tendam a se concentrar em torno de um certo valor intermediário. Estes erros se comportam de maneira aparentemente aleatória.

Algumas das principais fontes de erros são: erros nos dados de entrada, erros no estabelecimento do modelo matemático, erros de arredondamentos durante a computação, erros de truncamentos, e por fim, erros humanos e de máquinas.

Erro Absoluto e Erro Relativo

Quando se calcula um resultado por aproximação, é preciso saber como estimar ou delimitar o erro cometido na aproximação, pois sem isso, a aproximação obtida fica sem significado. Para delimitar o erro, são utilizados dois conceitos: o **erro absoluto** e o **erro relativo**.

Seja x' um valor aproximado para uma quantidade cujo valor exato é x . O erro absoluto em x' é dado por:

$$\epsilon_{x'} = |x' - x|, \quad (\text{A-1})$$

$$-\epsilon_{x'} \leq x' - x \leq \epsilon_{x'},$$

$$x' = x \pm \epsilon_{x'}. \quad (\text{A-2})$$

A equação A-2 diz que no cálculo de x' , o maior erro pode ser $+\epsilon_{x'}$ e o menor erro pode ser $-\epsilon_{x'}$.

O erro relativo em x' é dado por:

$$\left| \frac{(x' - x)}{x} \right|. \quad (\text{A-3})$$

O erro relativo geralmente é dado como uma porcentagem.

Margem de Erro

A margem de erro diz respeito a com que certeza as afirmações estão sendo feitas. A margem de erro pode ser entendida como a distância tolerável entre o valor “verdadeiro” e a estimativa. Esta medida é necessária pois, como a amostra é apenas um conjunto restrito e não toda a população, dá-se um limite máximo à incorreção que pode ocorrer, e com isso esta imprecisão pode ser controlada.

Os valores que podem ser utilizados para margem de erro são um balanceamento ou precisão feitas com bom senso, pois não há convenção ou prescrição literária que forneça estes valores.

A diferença entre erro amostral e margem de erro é que o erro amostral é calculado a partir das informações coletadas em um trabalho já realizado, já a margem de erro é estipulada na fase do cálculo do tamanho da amostra para um trabalho a realizar. Geralmente o erro amostral é menor ou igual à margem de erro.

Média Aritmética

A **média aritmética** de um conjunto de dados é uma medida de sua localização, ou tendência central, é simplesmente a soma de todos os valores, dividida pela quantidade total de elementos do conjunto.

Média Amostral

A **média amostral** de um conjunto de dados é uma medida de sua localização, ou tendência central, é simplesmente a soma de todas as amostras possíveis que podem ser retiradas de uma população, dividida pela quantidade total de amostras do conjunto. O valor da média amostral será representado por $\bar{x}$.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (\text{A-4})$$

onde x_i é o i -ésimo valor e n é o número total de valores na amostra.

Intervalo de Confiança

Os resultados de uma amostragem são expressos por **intervalos de confiança**. Como não se sabe o valor exato do parâmetro populacional por meio de uma amostra, o intervalo de confiança é dado por dois valores, chamados de Limite Inferior e Limite Superior do Intervalo de Confiança.

Níveis de Confiança

O **nível de confiança** é definido por:

$$P(\theta_1 \leq \theta \leq \theta_2) = 1 - \alpha, \quad (\text{A-5})$$

sendo que:

θ : parâmetro populacional de interesse;

θ_1 : limite inferior de um intervalo de confiança montado para estimar θ ;

θ_2 : limite superior de um intervalo de confiança montado para estimar θ ;

α : nível de significância (100% - nível de confiança);

$1 - \alpha$: nível de confiança. Exemplo: se o nível de confiança desejado é 95% então

$$1 - \alpha = 0,95.$$

Variância Amostral

A variância amostral é dada por:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (\text{A-6})$$

onde x_i é o i -ésimo valor, n é a quantidade total de valores da amostra, e $\bar{x}$ é a média amostral. Note que a variância é uma espécie de média dos quadrados dos desvios, com a diferença que o denominador não é a quantidade total de observações (n), e sim $n - 1$.

Desvio Padrão Amostral

O desvio padrão é uma medida de espalhamento das observações em torno da média. Primeiro calcula-se a diferença, ou **desvio**, de cada valor individual em relação à média amostral:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (\text{A-7})$$

A divisão por $n - 1$ aparece quando exige-se que a variância amostral s^2 seja um estimador não tendencioso da variância populacional σ^2 ¹.

O desvio padrão geralmente é usado para definir intervalos em torno da média.

¹ A variância populacional é dada por $\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$

Erro Padrão na Média

O erro padrão da média é dado pela razão entre o desvio padrão e a raiz do tamanho da amostra:

$$\sigma_{\bar{x}} = \frac{s}{\sqrt{n}}. \quad (\text{A-8})$$

Covariância e Correlação

Sejam duas variáveis aleatórias x e y . Quando ocorrem altos valores de x e de y ao mesmo tempo, diz-se que as duas variáveis aleatórias apresentam uma certa covariância², ou seja, uma tendência de se desviarem de forma parecida em relação às respectivas médias. A medida de covariância é obtida a partir do produto dos desvios $(x_i - \bar{x})$ e $(y_i - \bar{y})$ para cada elemento da amostra. O valor numérico da covariância é por definição a média dos produtos dos desvios:

$$\text{Cov}(x, y) = \frac{1}{n-1} \sum (x_i - \bar{x})(y_i - \bar{y}), \quad (\text{A-9})$$

onde (x_i, y_i) são os valores das observações individuais para o elemento i , $(\bar{x}, \bar{y})$ são as médias amostrais e n é a quantidade total de elementos na amostra. Note que $\text{Cov}(x, x)$ é a própria covariância de x .

Como o valor da covariância depende da escala usada para medir x e y , é difícil usá-la como padrão para comparar o grau de associação estatística de diferentes pares de variáveis. Para eliminar este problema, aplica-se um valor de escala, dividindo cada desvio individual pelo desvio padrão da variável correspondente. Este fator é chamado de **coeficiente de correlação** das duas variáveis, dado por:

$$r(x, y) = \frac{1}{n-1} \sum \left(\frac{x_i - \bar{x}}{s_x} \right) \left(\frac{y_i - \bar{y}}{s_y} \right). \quad (\text{A-10})$$

Pela definição, o coeficiente de correlação de qualquer par de valores é sempre um número que está no intervalo $[-1, +1]$. As correlações de diferentes pares de variáveis passam a ser medidas na mesma escala e podem então ser comparadas diretamente. Um coeficiente de correlação nulo significa apenas que uma relação linear não está presente. Podem haver outros tipos de dependência que não sejam refletidos pelo valor numérico de coeficiente de correlação. Variáveis ligadas por uma correlação linear perfeita tem coeficiente de correlação igual a $+1$. Valores intermediários apresentam relações parcialmente lineares, e o valor numérico do coeficiente de correlação é muito usado em trabalhos científicos como argumento a favor da existência de uma relação entre duas variáveis.

²Co-variável significa variar junto.

A.4 Métodos Estatísticos para Análise

Nesta seção será dada uma breve explicação sobre alguns métodos estatísticos: primeiro, a Análise da Variância de um Fator, que faz parte da inferência estatística, Subseção A.4.1, o Fatorial Completo, que possibilita fazer uma análise multivariada, Subseção A.4.2 e por último, a técnica chamada Quadrado Latino Subseção A.4.3.

A.4.1 Análise de Variância de um Fator

Na maioria dos experimentos, o interesse é comparar várias condições das variáveis independentes e analisar se existem diferenças entre elas. Uma abordagem muito utilizada quando se quer comparar várias médias é a **análise de variância**. A chamada **Análise de Variância Univariável** (*Single-Factor Analysis of Variance - ANOVA*), é um método que permite comparar simultaneamente populações para determinar se elas são idênticas ou significativamente diferentes. A ideia básica de ANOVA é assumir que toda variação não aleatória nas observações experimentais são devido às diferenças de desempenho médio nos níveis alternativos dos fatores experimentais.

Suponha que existam a **tratamentos**, ou **diferentes níveis** ou um **único fator** que se deseja comparar. A resposta observada de cada tratamento a é uma variável aleatória, que pode ser discreta ou contínua. Os dados podem ser representados como na tabela A.1. Uma entrada y_{ij} representa a j -ésima observação feita sobre o fator em um nível de tratamento i . Haverá, em geral, n observações sobre o i -ésimo tratamento.

Tabela A.1: *Dados típicos de um Experimento com um Fator.*

Tratamento (Nível)	Observações				Totais	Médias
1	y_{11}	y_{12}	$\cdots$	y_{1n}	y_1	$\bar{y}_1$
2	y_{21}	y_{22}	$\cdots$	y_{2n}	y_2	$\bar{y}_2$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$
a	y_{a1}	y_{a2}	$\cdots$	y_{an}	$\frac{y_a}{y}$	$\frac{\bar{y}_a}{\bar{y}}$

Para descrever as observações de experimento, um modelo é muito útil. Ele pode ser descrito por:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij}, \quad (\text{A-11})$$

em que:

μ = média global, parâmetro comum a todos os tratamentos;

τ_i é um parâmetro característico do i -ésimo tratamento, conhecido como efeito do i -ésimo tratamento.

$$\mu = \frac{\sum_{i=1}^a \mu_i}{a}, \quad (\text{A-12})$$

$$\tau = \sum_{i=1}^a \tau_i = 0; \quad (\text{A-13})$$

pelo fato de $\mu_i = \mu + \tau_i$, e μ é uma constante e τ_i representa desvios dessa constante quando os tratamentos específicos são aplicados.

Este modelo para cálculo de médias é um **modelo linear estatístico**, isto é, a variável y_{ij} é uma função linear dos parâmetros do modelo. A equação A-11 é também chamada de análise de variância de um fator porque somente um fator é investigado. O experimento deve ser realizado em um ambiente aleatório para que os tratamentos sejam aplicados de uma maneira uniforme. Desta forma, o experimento é chamado de **modelo completamente aleatório**. Os objetivos são testar hipóteses apropriadas sobre o tratamento e estimá-las.

O modelo que considera os tratamentos com variáveis fixas é chamado de **modelo de efeitos fixos**, e o que considera variáveis aleatórias é chamado de **modelo de efeitos aleatórios** ou **modelo de componentes de variância**.

Rardin e Uzsoy [95] explicam como foi modelado um experimento com ANOVA, fazendo sempre a relação com algoritmos. Portanto, seja um experimento computacional com níveis dos parâmetros problema $p = 1, \dots, P$ e algoritmos $a = 1, \dots, A$, as variáveis resposta podem ser assumidas como

$$y_{pak} = \mu + \tau_p + \beta_a + \epsilon_{pak}, \quad (\text{A-14})$$

onde y_{pak} é a variável resposta sobre $k \in \{1, \dots, k\}$ replicações de testes no nível p do parâmetro problema e no nível a do parâmetro algoritmos. A variável μ equivale a média geral; τ_p é o efeito incremental do parâmetro problema no nível p assumindo que $\sum_p \tau_p = 0$; β_a é o efeito incremental do algoritmo a assumindo que $\sum_a \beta_a = 0$; e ϵ_{pak} é o erro aleatório da observação pak . A técnica de blocagem não é utilizada, com a resolução das mesmas instâncias por todos os algoritmos.

ANOVA estima cada uma das várias médias e particiona a soma total dos quadrados, ou desvio quadrado da média da amostra global em partes separadas devido a cada fator experimental e ao erro. Por exemplo, para o modelo A-14:

$$SS_{total} = \sum_{p=1}^P \sum_{a=1}^A \sum_{k=1}^K (y_{pak} - \bar{y}_{...})^2 \quad (\text{A-15})$$

$$= A \sum_{p=1}^P (\bar{y}_{p..} - \bar{y}_{...})^2 + P \sum_{a=1}^A (\bar{y}_{.a.} - \bar{y}_{...})^2 + \sum_{p=1}^P \sum_{a=1}^A \sum_{k=1}^K (y_{pak} - \bar{y}_{...} - \bar{y}_{.a.} + \bar{y}_{p..})^2 \quad (\text{A-16})$$

$$= SS_{problems} + SS_{algorithms} + SS_{error}, \quad (\text{A-17})$$

onde a barra denota as médias amostrais e os pontos indicam as médias que foram calculadas. Depois de dividir pelo grau de liberdade em cada efeito, a média quadrada resultante indica a importância relativa dos vários fatores para explicar os resultados experimentais.

Além do fato da maioria dos experimentos computacionais terem mais que um fator, duas extensões surgem em muitos casos. Primeiro, não é incomum encontrar **interações** entre fatores. Isto é, por exemplo, o efeito do algoritmo a poder variar com o nível do fator p . Pois somente o problema ou algoritmo não permitem inferir todas as conclusões. Algumas interações são feitas em ANOVA introduzindo uma nova variável y_{pa} para cada combinação pa ao modelo A-14, para obter:

$$y_{pak} = \mu + \tau_p + \beta_a + \gamma_{pa} + \epsilon_{pak}, \quad (\text{A-18})$$

com $\sum_p \gamma_{pa} = 0$ para cada a e $\sum_a \gamma_{pa} = 0$ para cada p .

A outra extensão quase sempre necessária na análise de experimentos com heurísticas é o ajuste para blocagem sobre as instâncias. A técnica de blocagem em modelos ANOVA é utilizada introduzindo outro termo linear ϕ_{pk} para cada instância k , em cada nível p do parâmetro problema. Por exemplo, o modelo A-18 resulta em:

$$y_{pak} = \mu + \tau_p + \beta_a + \gamma_{pa} + \phi_{pk} + \epsilon_{pak}, \quad (\text{A-19})$$

adicionando $\sum_p \phi_{pk} = 0$ para todo p . As k instâncias tornam-se um novo fator “instância” no experimento que é agrupado, pois não pode ser comparado entre os níveis dos p fatores.

Até o momento, na análise de variância é assumido que a variação não aleatória é devido somente à diferenças nas médias e nos efeitos adicionados (equações A-14 a A-19).

A normalidade e a variância do erro comum estão geralmente associadas com ANOVA quando se tenta atribuir significância estatística para saber se uma média quadrada é grande em relação à média dos erros quadrados.

Mas há razão para duvidar, por vezes, sobre o teste de significância de ANOVA no contexto da experimentação heurística. Por exemplo, o tamanho do problema é muitas vezes um fator nos testes com heurísticas. Os resultados podem ser consistentes se o tamanho do problema aumenta. De qualquer maneira a variância do erro não é constante

nos níveis de tamanhos diferentes.

Quando são feitos testes de significância e existe dúvida sobre a suposição da variância comum, a solução mais utilizada é transformar em respostas experimentais. Uma maneira eficaz de fazer isso é usando *ranks*, isto é, substituir a melhor resposta por 1, a segunda por 2, etc, com *ranks* duplicados permitido respostas iguais.

No contexto de heurística, o *ranking* geralmente é baseado na qualidade da solução e na porcentagem de erro. Resultados de cada instância do problema são classificados com aumento de uma medida (para um problema de minimização), com o melhor algoritmo sobre a instância 1, o segundo mais efetivo como 2, etc. Desde que a variação entre a sequência de inteiros seja previsível, o efeito é estabilizar a variação dentro dos blocos de instâncias. Então executar ANOVA sobre os números *rank* pode produzir mais indicações confiáveis da significância do fator.

Mais detalhes de como aplicar o ANOVA são encontrados em [75].

A.4.2 Fatorial Completo

Há uma alternativa para os testes comparativos, que é praticada pelas ciências empíricas desde a época de Francis Bacon. É a chamada experimentação controlada. Por exemplo, um bom resultado de um experimento depende de algumas características, que são encontradas fazendo um experimento controlado que verifica a presença ou ausência das características que afetam o desempenho do algoritmo.

Um dos problemas mais comuns na execução de experimentos é determinar a influência de uma ou mais variáveis sobre uma outra variável de interesse. Por exemplo, suponha que um determinado problema está sendo resolvido com *Simulated Annealing*, e objetiva-se saber qual será a variação no valor de uma solução ótima encontrada ao variar o valor da taxa de recozimento e a solução inicial. Em outras palavras, se quer descobrir como a solução depende dos fatores taxa de recozimento e solução inicial. Este problema pode ser visto como um problema particular do problema mostrado na Figura A.1. Um certa quantidade de fatores $F_1, F_2, \dots, F_n$ atuando num sistema em estudo, produz as respostas $R_1, R_2, \dots, R_n$. O sistema atua como uma função que opera sobre as variáveis de entrada e produz como saída as respostas observadas [10].

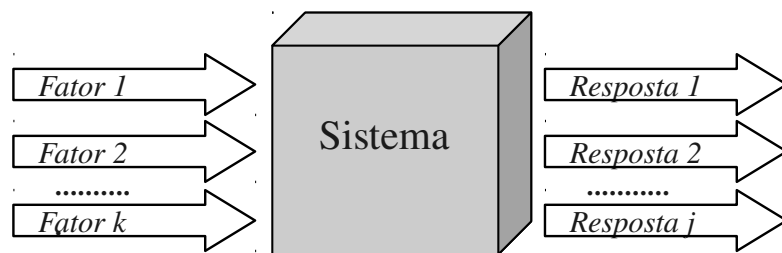


Figura A.1: Um sistema representado por uma função ligando os fatores (variáveis de entrada) às respostas (variáveis de saída). Retirado de [10].

O **Planejamento Fatorial de Dois Níveis**, é muito útil em investigações preliminares, quando se quer saber se determinados fatores têm ou não influência sobre a resposta. É um planejamento simples de executar depois pode ser ampliado para formar um planejamento mais sofisticado. Para utilizar este método, é importante fazer uma triagem para saber quais são os fatores que merecem um estudo mais profundo, pois muitos fatores podem atrapalhar os resultados, pois alguns deles talvez não tenham influência significativa sobre a resposta.

Para aplicar um planejamento fatorial deve-se primeiramente especificar os níveis em que cada fator deve ser estudado. Num **planejamento fatorial completo** realizam-se experimentos em todas as possíveis combinações dos níveis dos fatores.

Para estudar o efeito de qualquer fator sobre uma dada resposta, é necessário mudar o fator de nível, e a observação destas mudanças produz os resultados. Para tanto, é necessário ter pelo menos um fator que possua no mínimo dois níveis diferentes. Para k fatores (k variáveis controladas pelo experimentador), um planejamento completo de dois níveis exige a realização de $2 \times 2 \times \dots \times 2 = 2^k$ ensaios diferentes. Este planejamento é chamado de **planejamento fatorial 2^k** .

O modelo experimental **fatorial completo** começa com uma lista de n fatores que podem afetar o desempenho, como o tamanho do problema, existência da solução etc. Cada fator i tem vários níveis $k_i = 1, \dots, m_i$, e correspondem a diferentes tamanhos de problemas, medidas etc. Os níveis não necessariamente correspondem a valores em uma escala, como por exemplo, se o fator é uma estrutura do problema e os níveis denotam vários tipos desta estrutura. Um problema é gerado para cada célula ($k_i = 1, \dots, k_n$) em um vetor n -dimensional e a média do desempenho é medida para cada conjunto. A análise estatística pode checar se, por exemplo, o fator 1 tem um efeito significativo no desempenho quando os fatores restantes são mantidos constantes em qualquer nível do conjunto ($k_1, \dots, k_n$). Também é possível medir interações entre os fatores.

Como é perceptível, este modelo requer geração de número aleatórios, visto que ele produz uma grande quantidade de instâncias aleatórias, mas apesar disso este

modelo tem pouca semelhança com a geração aleatória tradicional. Pois a meta não é gerar problemas reais, e sim gerar vários conjuntos de testes (instâncias), onde cada um é homogêneo com relação às características que são susceptíveis de afetar o desempenho.

O trabalho de Balestrassi et al. [90] mostra uma aplicação do método fatorial completo em redes neurais.

A.4.3 Quadrado Latino

Segundo Montgomery [75], o modelo experimental **Quadrado Latino** é um modelo que utiliza o princípio de blocagem. Quando a fonte de perturbação de variabilidade é conhecida e controlável, o modelo é chamado blocagem, e é usada para eliminar sistematicamente seu efeito sobre comparações estatísticas entre tratamentos.

É usado para eliminar duas fontes de perturbação de variabilidade, isto é, permite sistematicamente fazer blocagem em duas direções. Em geral, um quadrado latino para p fatores (também chamado de $p \times p$), contém p linhas e p colunas. Cada célula contém uma das p letras que correspondem aos tratamentos, e cada letra ocorre somente uma vez em cada linha e coluna. O arranjo quadrado de tratamentos (ou formulações) foi denotado inicialmente por letras latinas, por isto o nome Quadrado Latino. Veja a Figura A.4.3, um exemplo de um quadrado latino de tamanho 4.

4 × 4			
A	B	C	D
B	C	A	D
C	D	B	A
D	A	C	B

Figura A.2: Exemplo de quadrado latino de tamanho 4

O modelo estatístico para um Quadrado Latino é:

$$y_{ijk} = \mu + \alpha_i + \tau_j + \beta_k + \epsilon_{ijk} \begin{cases} i = 1, 2, \dots, p; \\ j = 1, 2, \dots, p; \\ k = 1, 2, \dots, p; \end{cases} \quad (\text{A-20})$$

onde y_{ijk} é a observação da i -ésima linha e da j -ésima coluna para o k -ésimo tratamento, μ é a média total, α_i é a i -ésima linha, τ_j é o j -ésimo tratamento, β_k a k -ésimo coluna, e ϵ_{ijk} é o erro aleatório. Neste modelo, não há interação entre as colunas, linhas e tratamentos, porque só existe uma observação única em cada célula.

A análise da variância consiste no particionamento da soma total dos quadrados de $N = p^2$ observações em componentes para linhas, colunas, tratamentos e erros. Por exemplo:

$$SS_T = SS_{Linhas} + SS_{Colunas} + SS_{Tratamentos} + SS_E, \quad (A-21)$$

com o respectivos graus de liberdade:

$$p^2 - 1 = p - 1 + p - 1 + p - 1 + (p - 2)(p - 1). \quad (A-22)$$

Supondo que ε_{ijk} é $NID(0, s^2)$, s^2 é a variância. Cada soma dos quadrados do lado direito da equação A-21, dividindo por s^2 , é uma variável aleatória qui-quadrada distribuída independente.

Esta técnica pode ser útil em situações onde linhas e colunas representam fatores que se deseja estudar e investigar onde não estão as restrições aleatórias. Desta forma, três fatores (linhas, colunas e letras), com p níveis, podem ser investigados com somente p^2 execuções. O modelo assume que não há interação entre os fatores.

Itens Para avaliação de Experimentos

Os itens da Tabela [B.1](#) foram encontrados no trabalho de Crowder, Dembo e Mulvey, de 1979 [[19](#)], como uma forma de guiar a facilitar a condução de experimentos. Está dividida em cinco partes, de A e E: apresentação dos resultados, implementação, modelo experimental, relato dos resultados e conclusões. Cada parte é subdividida em itens recomendados e itens opcionais. O *checklist* desenvolvido neste trabalho usou como base estes dados.

Tabela B.1: *Checklist de relato de experimento computacional, retirado de Crowder, Dembo e Mulvey [19].*

Item	Recomendado	Opcional
A1 - Apresentação dos Algoritmos	a) Descrição completa do algoritmo b) Classe do problema (tipo e tamanho) que o algoritmo resolve.	1) Análise de complexidade 2) Teoremas de convergência 3) Análise da quantidade de trabalho por iteração
A2 - Implementação	a) Linguagem de programação b) Compilador e opções c) Ambiente computacional d) Descrição dos dados de entrada e) Descrição das configurações f) Condições em que o código está disponível g) Manual de instruções h) Descrição de técnicas de pré-processamento i) Descrição de outras técnicas (estratégia inicial)	
A3 - Modelo Experimental	a) Definição clara dos objetivos do experimento b) Documentação de novas instâncias c) Acurácia com que as soluções são obtidas d) Descrição do gerador de instâncias e) Descrição do pré-processamento da heurística f) Descrição completa do conjunto de instâncias de teste	
A4 - Relato dos Resultados	a) Justificativa das medidas utilizadas (tempo de CPU, iterações, robustez, etc) b) Para tempo de CPU: quantas vezes foi computado e se inclui entrada/saída; tempo de pré-processamento; métodos para padronizar resultados c) Detalhes de regras de parada e critério de convergência d) Descrição de quantas operações são feitas na função objetivo, iterações, etc. e) Requisitos das condições iniciais f) Solução final e valores das variáveis g) Para comparação de vários métodos: critério de convergência, tolerâncias, requisitos de armazenamento, condições iniciais, tempo de CPU, acurácia alcançada.	1) Resultados dos diferentes critérios de término, parâmetros, diferentes inicializações, etc.
A5 - Conclusões	a) Justificar as conclusões a partir dos dados apresentados	1) Direções nas pesquisas futuras, possíveis melhoras no algoritmo, identificação das instâncias que não foram resolvidas.