



CSBC2015

a internet de tudo, toda observada

**XXXV CONGRESSO DA SOCIEDADE BRASILEIRA DE
COMPUTAÇÃO**

**De 20 de Julho a 23 de Julho de 2015
Recife - PE**

**Anais da 34^a Jornada de Atualização
em Informática
JAI 2015**

Editora

Sociedade Brasileira de Computação (SBC)

Realização

Centro de Informática (CIn) da UFPE

Promoção

Sociedade Brasileira de Computação (SBC)

Agência Brasileira do ISBN

ISBN 978-85-88442-99-3



9 788588 442993

Sumário

Prefácio	4
Capítulo 1: <i>Teoria da Computação: uma introdução à Complexidade e à lógica computacional</i>	10
Capítulo 2: <i>Computação Urbana: Técnicas para o Estudo de Sociedades com Redes de Sensoriamento Participativo</i>	68
Capítulo 3: <i>Introdução à Otimização Combinatória</i>	123
Capítulo 4: <i>IFML: Linguagem de Modelagem de Fluxo de Interação</i>	190
Capítulo 5: <i>Uma introdução à complexidade parametrizada</i>	232
Capítulo 6: <i>Simulação de Robôs Móveis e Articulados: Aplicações e Prática</i>	274

Prefácio

Este livro contém os textos de referência para os minicursos da 34^a Jornada de Atualização de Informática promovida pela Sociedade Brasileira de Computação, em seu congresso anual, que em 2015 realiza-se na cidade de Recife, no Estado de Pernambuco, entre 20 e 24 de julho, com a organização geral do Centro de Informática da Universidade Federal de Pernambuco.

Esta 34^a edição da JAI nos regala com 6 minicursos, cada qual contemplado com um capítulo deste livro, que seguem a ordem dos minicursos, trazendo aos estudantes de graduação e pós-graduação, bem como aos profissionais e estudiosos da área, atualizações em teoria e sistemas de computação, com metade dos capítulos dedicados a cada uma das vertentes.

Os Capítulos 1 e 3 introduzem e trazem avanços nos fundamentos teóricos da ciência da computação, tais como classes de problemas, computabilidade e lógica computacional, além de técnicas de resolução de problemas difíceis (em otimização combinatória), apresentando conceitos, além de novos e velhos desafios que tem intrigado pesquisadores da área desde sempre. Por outro lado, o Capítulo 5 introduz conceitos para refinamentos na classificação de problemas difíceis, completando destarte juntamente com os Capítulos 1 e 3, um panorama extenso da diversidade do que se entende ser a área de teoria da computação, marcando a criação da comissão especial desta área na Sociedade Brasileira da Computação em 2014.

Em contraste com os capítulos anteriormente citados, que tem ênfase em seus aspectos teóricos, nos Capítulos 2 e 6 a ciência da computação se apresenta em algumas de suas vertentes mais arrojadas, pelas áreas de computação urbana e robótica móvel, onde o primeiro capítulo, além de introduzir o conceito de computação urbana, discorre sobre as técnicas de sensoriamento das redes urbanas; e o segundo apresenta os principais conceitos de robótica móvel e de robôs articulados. O Capítulo 6, por sua vez, tem foco em aspectos práticos de engenharia de software, introduzindo uma linguagem de especificação para projeto de interface de sistemas de informação e exemplificando o uso da mesma.

Essa coleção de temas aqui apresentados, tão diversos, permitirão aos leitores apreciar a riqueza da área da ciência da computação e o seu potencial quase inesgotável de apresentar desafios e soluções, teóricas e práticas, visando contribuir para a melhoria e bem estar das sociedades modernas.

Os temas por si só já seriam suficientes para fazer desta 34^a mais uma edição memorável da JAI. O fato porém de que entre os autores dos capítulos encontram-se alguns de nossos mais experientes e renomados pesquisadores, ao lado de jovens e talentosos pesquisadores, que demonstraram destreza e habilidade excepcionais para introduzir temas de difícil abordagem, traz a esta 34^a edição um brilho especial. Faz-se necessária portanto uma menção de agradecimento especial a todos os autores pela grande contribuição que fazem à nossa Sociedade Brasileira de Computação, prestigiando-nos com as suas participações.

Antes de fazer uma breve apresentação dos autores, é preciso enaltecer e agradecer o empenho e o excelente trabalho colaborativo do comitê científico desta 34^a edição. Ressalte-se que o pronto

apoio da organização geral do CSBC 2015, bem como da Diretoria da SBC proveu todas as condições para a realização desta edição da JAI. Finalizamos desejando a todos os participantes, autores, palestrantes e público, um excelente CSBC e uma JAI mais do que proveitosa!

Cláudia Linhares Sales
Coordenação Geral da 34^a JAI

Henrique Rebêlo
Coordenação Local da 34^a JAI

Breve Apresentação dos Autores

JAI 1 – Teoria da Computação: Uma Introdução à Complexidade e à Lógica Computacional

Celina Miraglia Herrera de Figueiredo obteve bacharelado (1982) e mestrado (1984) em Matemática na PUC-Rio, mestrado (1987) em Matemática no UMIST (UK), doutorado (1991) em Engenharia de Sistemas e Computação na COPPE/UFRJ com período sanduíche na University of Waterloo, Canadá. Fez carreira docente na UFRJ, onde ingressou no Instituto de Matemática em 1989, e na COPPE em 1991. Fez pós-doutorado em 1995 na University of Waterloo, Canadá. Atualmente é professora titular do Programa de Engenharia de Sistemas e Computação da COPPE, onde coordena o Núcleo de Excelência em Algoritmos Randomizados, Quânticos, e Aproximativos: Projeto, Análise e Implementação de Soluções Eficientes para problemas Combinatórios Fundamentais. É pesquisadora na área de Ciência da Computação, com ênfase em Teoria da Computação, e lidera o grupo de algoritmos e combinatória da COPPE, atuando principalmente nos seguintes temas: teoria dos grafos, algoritmos e complexidade computacional. Tem bolsa de produtividade em pesquisa do CNPq desde 1992, estando atualmente no nível 1A. É desde 2005 Cientista do Nosso Estado FAPERJ. Recebeu em 2006 o Prêmio Giulio Massarani de Mérito Acadêmico da COPPE. Recebeu em 2013 homenagem na solenidade comemorativa dos 50 anos da COPPE.

Luís C. Lamb é PhD in Computer Science, Imperial College London, University of London (2000); é Mestre UFRGS (1995) e Bacharel em Ciências de Computação - UFRGS (1992). Realizou estágio pós- doutoral no Group of Logic, Language and Computation, King's College London (2005- 2006). Em 2010, concluiu o MIT Executive Program in Strategy and Innovation e em 2014 concluiu o Executive Program in Management and Leadership. Atualmente é Professor Titular do Departamento de Informática Teórica e Diretor do Instituto de Informática da UFRGS (2011 a 2015). É Honorary Visiting Fellow na City University London. É autor de dois livros científicos: 'Neural-Symbolic Cognitive Reasoning', Springer (2009) e 'Compiled Labelled Deductive Systems' sobre lógicas não-clássicas (IoP, 2004), além de diversos artigos em

conferências e periódicos de referência. Tem experiência nas áreas de Ciência da Computação e Lógica Aplicada. É co-editor-chefe da Revista de Informática Teórica e Aplicada, membro do 'Editorial Board' do Logic Journal of the Interest Group in Pure and Applied Logics, na subárea de 'Algorithms and Neural Networks' e membro do 'Editorial Board' do Journal of the Brazilian Computer Society. Atualmente é membro do Comitê Assessor de Ciência da Computação do CNPq e Bolsista de Produtividade em Pesquisa nível 1C do CNPq.

JAI 2 – Computação Urbana: Técnicas para o Estudo de Sociedades com Redes de Sensoriamento Participativo

Thiago Henrique Silva. Pesquisador no Departamento de Ciência da Computação da UFMG. Em 2011, esteve como pesquisador visitante na Telecom Italia, Itália, onde trabalhou na área de ambientes inteligentes. Durante o seu doutoramento, fez estágios na University of Birmingham, Reino Unido, e no INRIAParis, França. Ao longo de sua carreira acadêmica foi agraciado com os prêmios de melhor trabalho/menção honrosa nos seguintes eventos: IEEE International Conference on Cyber, Physical and Social Computing (2012), no Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (2009, 2010 e 2013), no Simpósio Brasileiro de Computação Ubíqua e Pervasiva (2012) e no concurso de aplicativos em dados abertos promovido pelo Ministério da Justiça (2013).

Antonio Alfredo Ferreira Loureiro. Bolsista de Produtividade em Pesquisa do CNPq Nível 1A. Possui graduação em Ciência da Computação pela Universidade Federal de Minas Gerais (1983), mestrado em Ciência da Computação pela Universidade Federal de Minas Gerais (1987) e doutorado em Ciência da Computação pela University of British Columbia, Canadá (1995). Atualmente é Professor Titular da Universidade Federal de Minas Gerais. Tem experiência na área de Ciência da Computação, com ênfase em Sistemas Distribuídos, atuando principalmente nos seguintes temas: algoritmos distribuídos, computação móvel/ubíqua, computação urbana, comunicação sem fio, gerenciamento de redes, redes de computadores, redes de sensores sem fio.

JAI 3 – Introdução à Otimização Combinatória

Flávio K. Miyazawa é professor titular do Instituto de Computação da Universidade Estadual de Campinas, UNICAMP, onde pesquisa sobre otimização combinatória, algoritmos de aproximação, algoritmos probabilísticos, teoria algorítmica dos jogos e projeto e análise de algoritmos combinatórios. Tem bolsa de produtividade em pesquisa do CNPq desde 1998, estando atualmente no nível 1B.

Cid Carvalho de Souza é professor titular do Instituto de Computação da Universidade Estadual de Campinas, UNICAMP, onde pesquisa sobre otimização combinatória, programação linear inteira, heurísticas e projeto e análise de algoritmos combinatórios. Tem bolsa de produtividade em

pesquisa do CNPq desde 1995, estando atualmente no nível 1C.

JAI 4 – IFML: Linguagem de Modelagem de Fluxo de Interação

Raul Sidnei Wazlawick possui Bacharelado em Ciência da Computação pela Universidade Federal de Santa Catarina (UFSC, 1988), Mestrado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS, 1990), Doutorado em Engenharia de Produção (UFSC, 1993) e Pós-Doutorado pela Universidade Nova de Lisboa (UNL, 1998). Desde 1992 é Professor da UFSC (atualmente como Associado IV). Criou e foi editor da Revista Brasileira de Informática na Educação da SBC entre 1997 e 2001. É autor dos seguintes livros: “Análise e Projeto de Sistemas de Informação Orientados a Objetos”, (Elsevier, 2004; 2a edição em 2011, 3a edição em 2015) “Metodologia de Pesquisa para Ciência da Computação” (Elsevier, 2009; 2a edição em 2014), “Engenharia de Software: Conceitos e Práticas” (Elsevier, 2012) e “Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML” (Morgan Kaufman, 2014). Tem experiência na área de Engenharia de Software, atuando principalmente em modelagem de sistemas orientados a objetos e melhoria do processo de desenvolvimento de software. No Ministério da Educação (MEC) foi membro do Comitê de Especialistas de Ensino em Ciência da Computação (CEEInf) por 4 anos e é atualmente membro da Comissão Assessora da Área de Sistemas de Informação para o ENADE. É Supervisor Geral do Projeto e-SUS, parceria da UFSC com o Ministério da Saúde para a reestruturação do sistema de informação da atenção básica (Postos de Saúde) no Brasil. Em 2013 recebeu o título de “Professor Honoris Causa” pelo Centro de InSTRUÇÃO de Guerra Eletrônica do Exército Brasileiro.

JAI 5 – Uma Introdução à Complexidade Parametrizada

Uéverton dos Santos Souza possui graduação em Tecnologia em Sistemas de Computação pela Universidade Federal Fluminense (2008), mestrado em Informática pela Universidade Federal do Rio de Janeiro (2010) e doutorado em Computação pela Universidade Federal Fluminense (2014). Atualmente é professor do ensino básico técnico e tecnológico do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca. Tem experiência na área de Ciência da Computação, com ênfase em Análise de Algoritmos e Complexidade de Computação, atuando principalmente nos seguintes temas: complexidade parametrizada, teoria dos grafos, pesquisa operacional, e otimização combinatória. A maioria dos trabalhos de pesquisa desenvolvidos pelo autor envolvem temas da teoria da complexidade parametrizada. Recebem uma citação especial os trabalhos em co-autoria com os professores Michael Fellows e Frances Rosamond, dois dos principais pesquisadores da área de complexidade parametrizada do mundo.

Vinicius Fernandes dos Santos possui graduação em Bacharelado em Ciência da Computação pela

Universidade Federal do Rio de Janeiro (2006), mestrado em Informática pela Universidade Federal do Rio de Janeiro (2009) e doutorado em Engenharia de Sistemas e Computação pela Universidade Federal do Rio de Janeiro (2013). Atua nos temas de algoritmos e teoria de grafos, principalmente na área de convexidade em grafos. Participou da Maratona de Programação como competidor, técnico e organizador. Atualmente é professor do Departamento de Computação do Centro Federal de Educação Tecnológica de Minas Gerais.

JAI 6 – Simulação de Robôs Móveis e Articulados: Aplicações e Prática

Fernando Santos Osório concluiu o doutorado em Informatique Systemes et Communications - Institut National Polytechnique de Grenoble (INPG/IMAG) em 1998. Atualmente é professor e pesquisador do Departamento de Sistemas de Computação do ICMC-USP (São Carlos-SP). É membro ativo da ACM, IEEE e SBC, onde na SBC foi membro do comitê gestor da CEIA (Comissão Especial de Inteligência Artificial 2007-2010) e foi membro do comitê gestor da CER (Comissão Especial de Robótica da SBC 2008-2014) e atualmente é o coordenador da CER (2014-2016). Em 2007 foi nomeado como membro do programa "Distinguished Visitors Program Latin America (DVP-LA 2007- 2009)" da IEEE Computer Society. Atua na área de Ciência da Computação, com ênfase em Inteligência Artificial e Robótica. Em suas atividades profissionais interagiu com cerca de 100 colaboradores em co-autorias de trabalhos científicos. Em seu currículo Lattes os termos mais frequentes na contextualização da produção científica, tecnológica e artístico-cultural são: Inteligência Artificial, Robótica Autônoma, Redes Neurais Artificiais, Aprendizado de Máquinas, Processamento de Imagens, Reconhecimento de Padrões, Realidade Virtual, Informática, Ambientes Virtuais e Computação Gráfica.

Rafael Alceste Berri possui bacharelado em Ciências da Computação (2005) e é mestre em Computação Aplicada (2014) pela Universidade do Estado de Santa Catarina. Atualmente é aluno do curso de Doutorado em Ciências de Computação e Matemática Computacional no ICMC/USP, onde já concluiu todos os créditos de disciplina do curso sendo aprovado com nota máxima em todas elas. Trabalhou como programador/analista de sistema por 12 anos, sendo 6 anos com software para Caldeiraria (CALDsoft Sistemas) e 6 anos no ERP interno da Malwee Malhas. Tem experiência na área de Ciência da Computação, com ênfase em Processamento Gráfico (Graphics), atuando principalmente nos seguintes temas: robótica e monitoramento do motorista (distrações ao volante). Possui 2 artigos publicados em conferências internacionais, 1 artigo em conferência nacional, 1 capítulo de livro aceito para ser publicado na CCIS Series pela Springer-Verlag, sendo todos estes trabalhos relacionados com aplicações em robótica. Atualmente é bolsista da CAPES e atua como estagiário no Programa PAE (Teaching assistant) junto a disciplina de graduação de SSC0712 – Programação de Robôs Móveis.

Comitê Científico

Cláudia Linhares Sales (UFC) - coordenadora
Alba Melo (UNB)
Bernadete Lóscio (UFPE)
David Deharbe (UFRN)
Jaime Sichman (USP)
Leonardo Sampaio (UECE)
Taisy Weber (UFRGS)
Thiago Noronha (UFMG)

Coordenação

Cláudia Linhares Sales (UFC) – coordenadora geral
Henrique Emanuel Mostaert Rebêlo (UFPE) – coordenador local

Capítulo

1

Teoria da Computação: Uma Introdução à Complexidade e à Lógica Computacional

Celina M. H. de Figueiredo e Luís C. Lamb

Universidade Federal do Rio de Janeiro e Universidade Federal do Rio Grande do Sul

Abstract

Computing theory is an abstract and mathematical research area, motivated by the challenges of computer science practice. Perhaps the main objective of theoretical computer science is to explore and understand the nature of computing, contributing to development of efficient and effective methodologies. This course introduces the basics of two fundamental subareas of theoretical computer science: computational complexity and computational logic. Such areas have significantly contributed to the development of computer science, not only from a foundational, but also from an applied perspective.

Resumo

A teoria da computação é uma área de pesquisa abstrata e matemática, e é motivada pelos desafios da prática da computação. O objetivo da teoria da computação é explorar e entender a natureza da computação, de modo a oferecer metodologias eficientes e corretas. Este curso apresenta uma introdução às duas subáreas fundamentais da teoria da computação: complexidade e lógica computacional. Estas duas áreas têm contribuído significativamente para o desenvolvimento da Ciência da Computação, não somente do ponto de vista fundamental, mas também sob uma perspectiva de aplicações.

Sumário

1.1 Uma Introdução à Complexidade -	
Celina M. H. de Figueiredo	12
1.1.1 O Problema do Milênio	13
1.1.2 O Guia para Problemas Desafiadores	19
1.1.3 Notas Bibliográficas	25
1.2 Lógica Para Computação: Uma Introdução Curta -	
Luís C. Lamb	30
1.2.1 Introdução: Brevíssimas Considerações Históricas	31
1.2.2 Lógica em Computação: Uma Breve Introdução	36
1.2.3 Lógica Proposicional	36
1.2.3.1 Exemplo de Sistema de Provas: Sistema Axiomático (<i>à la</i> Frege-Hilbert)	40
1.2.3.2 Exemplo de Sistema de Provas: Sistema de Dedução Natural	41
1.2.4 O Problema da Satisfatibilidade (SAT)	47
1.2.5 Lógica de Primeira Ordem	47
1.2.5.1 Dedução Natural para Lógica de Predicados	49
1.2.5.2 Outros Métodos de Prova	51
1.2.6 Lógicas-Não-Clássicas	52
1.2.6.1 Lógicas Não-Clássicas: Lógicas Modais	53
1.2.6.2 Semântica de Mundos Possíveis	56
1.2.6.3 Sistemas de Provas Para Lógicas Modais	57
1.2.6.4 Lógica em Inteligência Artificial: Integrando Raciocínio e Aprendizado	59
1.2.7 Conclusões e Perspectivas	63

1.1. Uma Introdução à Complexidade -

Celina M. H. de Figueiredo

Abstract

“To solve or to verify?” It is a question worth a million dollars. In 2000, the Clay Institute for Mathematics distinguished seven problems considered central to the progress of mathematics, calling them The Millennium Problems. The solution of each problem corresponds to a prize of one million dollars. One of the seven selected problems is a Theory of Computation problem: Is there a question whose answer can be quickly verified but the answer requires a long time to be found? This Millennium Problem, known as P versus NP, is the central problem in the Computational Complexity area where we try to rank the difficulty of the problems according to the efficiency of possible solutions through computer algorithms. We discuss the P versus NP problem, by classifying challenging problems in Polynomial or NP-complete and defining classes of problems that admit a dichotomy, where each problem in the class is classified as polynomial or NP-complete.

Resumo

“Resolver ou Verificar?” é uma pergunta que vale um milhão de dólares. No ano 2000, o Instituto Clay para Matemática distinguiu sete problemas considerados centrais para o progresso da matemática, chamando-os de Os Problemas do Milênio. A solução de cada problema corresponde a um prêmio de um milhão de dólares. Um dos sete problemas selecionados é um problema de Teoria da Computação: existe pergunta cuja resposta pode ser verificada rapidamente mas cuja resposta requer muito tempo para ser encontrada? Esse Problema do Milênio, conhecido como P versus NP, é o problema central na área de Complexidade Computacional, onde tentamos classificar a dificuldade dos problemas de acordo com a eficiência das possíveis soluções através de algoritmos computacionais. Discutiremos o problema P versus NP, através da classificação de problemas desafiadores em polinomial ou NP-completo e na definição de classes de problemas que admitem um resultado de dicotomia, onde cada problema da classe é classificado como polinomial ou NP-completo.

1.1.1. O Problema do Milênio

O computador é um aliado imprescindível para resolver os complexos problemas que surgem em biologia, química, física, economia, áreas nas quais pesquisadores se dedicam à modelagem e à simulação de problemas de larga escala com uso de computadores. Porém, há problemas que têm resistido à habilidade dos programadores. À medida que resolvemos problemas cada vez maiores e mais complexos por meio de enorme poder computacional e algoritmos engenhosos, os problemas resistentes e desafiadores ganham destaque. Nesse sentido, a teoria que propõe o problema do milênio ajuda a entender as limitações computacionais fundamentais. Uma questão de interesse teórico explica a dificuldade prática de problemas formulados em toda a comunidade científica.

No ano 2000, o Instituto Clay para Matemática distinguiu sete problemas considerados centrais para o progresso da matemática, chamando-os de Os Problemas do Milênio. A solução de cada problema corresponde a um prêmio de um milhão de dólares. Um dos sete problemas selecionados é um problema de Teoria da Computação: existe pergunta cuja resposta pode ser verificada rapidamente mas cuja resposta requer muito tempo para ser encontrada? Esse Problema do Milênio, conhecido como P versus NP, é o problema central na área de Complexidade Computacional, onde tentamos classificar a dificuldade dos problemas de acordo com a eficiência das possíveis soluções através de algoritmos computacionais [22].

O problema P versus NP pode ser informalmente resumido na pergunta “Resolver ou Verificar?”, que é ilustrada pela seguinte estória:

Em 1903, o matemático americano Frank Cole provou que o número $2^{67} - 1 = 147573952589676412927$ não é primo exibindo a fatoração $193707721 \times 761838257287$. É simples (embora tedioso se feito manualmente) calcular $2^{67} - 1$, calcular o produto $193707721 \times 761838257287$ e verificar que dão o mesmo número. Já encontrar essa fatoração é difícil. Cole disse que ele levou três anos trabalhando aos domingos.

“Resolver ou Verificar?” é uma pergunta que vale um milhão de dólares.

Duas campanhas: vacinação e pavimentação

Imagine que uma campanha de vacinação no estado do Rio de Janeiro precisa visitar cada uma das capitais dos seus 50 municípios. Por restrições de custo, a equipe responsável, saindo da cidade do Rio de Janeiro, precisa visitar cada uma das outras 49 cidades e retornar à cidade de partida, realizando um circuito que usa as rodovias do estado e que visita cada capital de município exatamente uma vez. Será que tal circuito pode

ser realizado no estado do Rio de Janeiro? Este circuito corresponde ao problema do Ciclo Hamiltoniano, formulado por William Hamilton em 1856, e conhecido também como o problema do Caixeiro Viajante. Uma condição suficiente para que tal circuito do vacinador exista é que cada cidade tenha muitas rodovias do estado que passem por ela mas esta condição não é necessária. Uma condição necessária é a inexistência de cidade gargalo, isto é, uma cidade tal que todas as rotas que conectam outras duas cidades usando rodovias do estado precisam passar por esta cidade gargalo mas esta condição não é suficiente. No estado do Rio de Janeiro, o município de Cabo Frio é um gargalo que isola o município de Búzios e impossibilita um circuito do vacinador. O município de Angra é outro gargalo e isola o município de Paraty.

Imagine agora que, ao invés de um vacinador que saindo da cidade sede da campanha visita num circuito um conjunto de cidades passando por cada cidade visitada uma única vez e retornando à cidade de partida, você considera uma campanha para recuperar a pavimentação das rodovias do estado. Por restrições de custo, a equipe responsável, saindo da cidade sede da campanha, realiza um circuito que percorre cada rodovia exatamente uma vez e retorna à cidade sede de partida. Será que tal circuito pode ser realizado no estado do Rio de Janeiro? Observe que o circuito do vacinador visita cada cidade do estado exatamente uma vez enquanto que o circuito do pavimentador percorre cada rodovia do estado exatamente uma vez. Possivelmente o vacinador deixa de percorrer alguma rodovia do estado enquanto que possivelmente o pavimentador visita uma cidade do estado mais de uma vez. O circuito do vacinador corresponde a uma permutação do conjunto das cidades do estado enquanto que o circuito do pavimentador corresponde a uma permutação do conjunto das rodovias do estado. Ambos problemas buscam a solução, o circuito desejado, num espaço com um número enorme de circuitos viáveis, da ordem de $50!$ circuitos, onde consideramos todas as possíveis permutações. É impraticável resolver o problema através de um algoritmo de força bruta que testa cada uma das permutações candidatas. Buscamos uma propriedade matemática do problema que reduza drasticamente este espaço exponencial de busca. Leonhard Euler resolveu definitivamente o problema do pavimentador ao publicar em 1736 o que consideramos o primeiro artigo científico da área de Teoria dos Grafos. Neste artigo, foi apresentada uma propriedade que caracteriza, uma propriedade que é ao mesmo tempo necessária e suficiente, para a existência de tal circuito do pavimentador: o número de rodovias que passam por cada cidade é par. Euler reduziu drasticamente o espaço exponencial de busca constituído de todas as possíveis permutações das rodovias. A poderosa condição de Euler fornece a solução após a fácil verificação dos graus das cidades.

Em Matemática e em Computação, a Teoria dos Grafos é a área que estuda estruturas matemáticas que modelam relações binárias entre objetos de um conjunto. Um grafo é definido por um conjunto de objetos chamados vértices e um conjunto de arestas que ligam pares de vértices. Euler em 1736 modelou o problema do pavimentador

através de um problema em Teoria dos Grafos: cada cidade corresponde a um vértice e cada rodovia entre duas cidades é representada por uma aresta que liga os dois vértices correspondentes. No estado do Rio de Janeiro temos 50 cidades capitais de municípios correspondendo a 50 vértices e temos rodovias que correspondem às arestas conectando estes vértices. A cidade capital de Itaguaí tem 7 rodovias que passam por ela, e corresponde a um vértice de grau ímpar no grafo das cidades capitais e rodovias do estado, o que segundo o Teorema de Euler impossibilita a existência de um circuito do pavimentador no estado. Por outro lado, caso encontremos um outro estado onde todas as cidades tenham grau par, esta condição fácil e poderosa de Euler garante que tal estado admite um circuito do pavimentador.

Para o problema do pavimentador, podemos decidir se o circuito existe ou não através da condição simples encontrada por Euler: basta conferir a paridade de cada cidade. Para o problema do vacinador, temos algumas condições apenas necessárias, temos outras condições apenas suficientes, mas ainda não temos uma condição que caracterize e resolva o problema, e não sabemos se tal condição existe! Para o problema do vacinador, hoje nos resignamos a buscar a resposta examinando o enorme espaço exponencial de busca constituído de todas as possíveis permutações das cidades!

Por outro lado, caso alguém persistente ou com muita sorte apresente um circuito do vacinador, um circuito que visita cada cidade exatamente uma vez, é fácil verificar que este circuito satisfaz a restrição: basta conferir que cada cidade é visitada uma vez e que cidades consecutivas no circuito são de fato conectadas por uma rodovia.

Será que o problema do vacinador é mais difícil que o problema do pavimentador? Será que existem problemas cuja solução pode ser verificada facilmente mas cuja solução não pode ser encontrada facilmente? Este é o desafio central para a Teoria da Computação.

Problemas P, NP e NP-completo

Vejamos novo exemplo. Considere uma turma em um colégio, digamos com 40 alunos, cada qual com certo número de amigos dentro da turma. O problema do Emparelhamento procura dividir aqueles alunos em 20 pares, cada par formado por alunos que sejam amigos. Note que o número de conjuntos contendo 20 pares de alunos quaisquer, não necessariamente amigos, é igual a $40!/(20!2^{20})$, aproximadamente 2^{78} , ou $3,2 \times 10^{23}$, um número de 24 algarismos! Segundo essa estimativa, temos algo como 3 mil vezes mais maneiras distintas de dividir nossa turma em 20 pares de alunos do que grãos de areia no mundo. O problema do emparelhamento é um problema fundamental na área de complexidade computacional. É um problema cuja solução sugeriu a definição formal adotada para computação eficiente. Um algoritmo é eficiente se a sua

execução consome um número de passos que cresce como um polinômio no tamanho dos dados da entrada. O problema do emparelhamento admite algoritmo que o resolve em n^3 passos, para uma turma de n alunos. Para n igual a 40, observe a drástica redução do exponencial 2^{78} , muito maior que 2^{40} , para o polinomial 40^3 , que dá um número de passos, ou operações básicas, próximo a 64 mil. Usamos o conceito de polinomial como sinônimo de tratável, viável, eficiente, em contraste com exponencial, sinônimo de inviável.

Considere algumas variações do problema do emparelhamento: o problema de se dividir a turma em grupos contendo 3 amigos mútuos em cada grupo; o problema de se dividir a turma em 3 grupos contendo apenas amigos mútuos em cada grupo; problema de organizar os alunos numa única mesa redonda de forma que apenas amigos sentem-se lado a lado (aqui reencontramos disfarçado o problema do vacinador, na turma de 40 alunos, existem $40!$ alocações possíveis numa mesa redonda); No caso de cada uma das variações do problema, ainda não conhecemos algoritmo que executa um número polinomial de passos. Cada uma das variações do problema define um problema matemático desafiador, tema de pesquisa avançada na área da Matemática chamada Combinatória. O desafio é encontrar alguma propriedade que corresponda a um atalho polinomial para buscar rapidamente no espaço exponencial das possíveis soluções. Estes problemas desafiadores compartilham uma propriedade: dada uma candidata a solução — um emparelhamento em grupos de três alunos compatíveis, uma alocação numa única mesa redonda, uma partição em três grupos de alunos compatíveis — podemos aprovar a candidata através de um algoritmo.

A questão central para Computação é: quão eficientemente um problema pode ser resolvido através de um algoritmo? Do ponto de vista computacional, distinguimos problemas fáceis e difíceis usando o conceito de algoritmo polinomial. Consideramos fáceis os problemas que podem ser resolvidos através de um algoritmo cujo número de passos cresce com uma potência fixa do número de símbolos usados para especificar a entrada do problema. Por outro lado, consideramos difíceis os problemas para os quais qualquer possível algoritmo consome um número extremamente grande de passos até retornar a resposta. Muitos problemas candidatos a problemas difíceis compartilham a propriedade: encontrar a solução parece difícil mas verificar a solução é fácil. A intuição diz que encontrar a solução tem que ser mais difícil do que simplesmente verificar a solução mas nem sempre a intuição é um bom guia para a verdade. Para estes problemas candidatos a problemas difíceis, não conhecemos ainda algoritmos polinomiais e não sabemos provar a inexistência de algoritmo polinomial. A Teoria da Complexidade Computacional considera estes problemas desafiadores, estes problemas que resistem à classificação em fácil ou difícil, como o problema do vacinador, dentro de uma única classe contendo problemas igualmente difíceis, igualmente desafiadores.

Em Teoria da Computação, chamamos de P a classe dos problemas que podem

ser resolvidos através de um algoritmo polinomial, um algoritmo cujo número de passos seja uma potência fixa no tamanho dos dados do problema. E chamamos de NP a classe dos problemas onde toda candidata a solução pode ser aprovada ou rejeitada em tempo polinomial. A condição de verificar parece bem menos restritiva que a de encontrar a solução. A igualdade $P = NP$ significaria que todo problema que tem solução que pode ser verificada rapidamente tem também solução que pode ser encontrada rapidamente. Para estudar a possível igualdade $P = NP$, os pesquisadores definiram o conjunto dos problemas igualmente difíceis entre si, e pelo menos tão difíceis quanto qualquer problema em NP. São os chamados problemas NP-completo. O problema do vacinador é um problema NP-completo. Caso alguém consiga um algoritmo de tempo polinomial para resolver o problema do vacinador, terá na verdade revolvido um problema que vale 1 milhão de dólares porque terá provado que $P = NP$.

Problemas desafiadores em Teoria dos Grafos

Um grafo é *perfeito* quando todo subgrafo induzido admite uma coloração própria de seus vértices (i.e., vértices adjacentes têm cores diferentes) que usa o mesmo número de cores que o número de vértices de uma de suas cliques (um conjunto de vértices mutuamente adjacentes). Os *grafos perfeitos* definem um rico tópico de pesquisa que pode ser visto como a interseção de três áreas: Teoria dos Grafos, Complexidade Computacional e Otimização Combinatória.

Claude Berge [5] introduziu a classe dos grafos perfeitos no início dos anos 1960, como uma classe de grafos interessante sob três aspectos, cada um com um problema aberto desafiador: uma classe definida a partir dos problemas de otimização combinatória NP-difíceis COLORAÇÃO MÍNIMA e CLIQUE MÁXIMA; uma classe cujo problema de reconhecimento (i.e., decidir se um dado grafo é perfeito) não se sabia sequer classificar como em NP; uma classe que sugeria uma caracterização por uma família auto-complementar infinita de subgrafos induzidos proibidos. Grötschel, Lovász e Schrijver [24] apresentaram em 1979 um algoritmo polinomial para a coloração mínima dos grafos perfeitos usando o método do elipsóide para programação linear, e deixaram em aberto o problema de encontrar um algoritmo polinomial de natureza puramente combinatória para colorir otimamente os grafos perfeitos. Johnson em 1981 na sua coluna *The NP-completeness – an ongoing guide* [28] propôs o problema de reconhecimento para a classe dos grafos perfeitos. Cameron [8] provou em 1982 que esse problema estava em CoNP. A prova de que a caracterização por subgrafos proibidos proposta por Berge estava correta e o algoritmo polinomial para o reconhecimento dos grafos perfeitos só apareceriam respectivamente em 2006 e em 2005 [12, 11]. O algoritmo polinomial de natureza puramente combinatória para colorir otimamente os grafos perfeitos ainda permanece como um desafio [37]. Descrevemos a seguir como se deu a classificação da

complexidade computacional de dois problemas centrais, abertos por décadas, um em polinomial e o outro em NP-completo.

Partição Assimétrica O problema da partição assimétrica foi proposto por Chvátal em 1985 no contexto de decomposições para grafos perfeitos [13]: dado um grafo G , o seu conjunto de vértices admite uma partição em quatro partes não vazias A, B, C, D tal que todo vértice em A é adjacente a todo vértice em B e todo vértice em C é não-adjacente a todo vértice em D ? Chvátal propôs a partição assimétrica como uma generalização de decomposições bem estudadas: corte estrela, corte clique e conjunto homogêneo, e acertou ao afirmar que a partição assimétrica teria um papel determinante na prova da conjectura de Berge para grafos perfeitos. No grafo G , o conjunto de vértices $A \cup B$ é um corte cuja remoção desconecta as partes não vazias C e D , e no complemento do grafo G , o conjunto de vértices $C \cup D$ é um corte cuja remoção desconecta as partes não vazias A e B . Dessa propriedade auto-complementar, vem o nome partição assimétrica, o conjunto $A \cup B$ é chamado *corte assimétrico*, e a partição A, B, C, D é chamada *partição assimétrica*.

O primeiro algoritmo polinomial para testar se um grafo admite uma partição assimétrica, um algoritmo recursivo de complexidade $O(n^{100})$, na verdade resolve um problema mais geral, a partição assimétrica com listas [15]. A versão com listas de um problema de partição tem como entrada, além do grafo cujo conjunto de vértices será particionado, uma lista de partes permitidas, para cada um dos vértices. Na versão com listas, as partes podem ser vazias. Muitos problemas combinatórios em grafos podem ser formulados como um problema de partição dos vértices em subconjuntos de acordo com restrições internas ou externas em relação às adjacências: conjunto homogêneo, corte clique, coloração de vértices, grafo *split*, entre outros. Feder, Hell, Klein e Motwani descreveram uma dicotomia para partições em quatro partes na versão com listas, onde classificaram cada um dos problemas dessa classe em quasi-polynomial ou NP-completo [14]. Um algoritmo de tempo quasi-polynomial tem complexidade de pior caso $O(n^{c \log^t n})$, onde c e t são constantes positivas e n é o número de vértices do grafo de entrada. O problema da partição assimétrica foi classificado nessa ocasião como quasi-polynomial, uma indicação de que o problema era de fato polinomial. Posteriormente, o algoritmo polinomial que desenvolvemos para partição assimétrica estabeleceu a complexidade do problema de Chvátal. O nosso algoritmo recursivo para partição assimétrica com listas recursivamente define problemas de partição assimétrica com listas menores. O número de subproblemas $T(n)$ encontrados satisfaz recorrências da forma $T(n) \leq 4T(9n/10)$, o que fornece um tempo de execução $O(n^{100})$, um desafio para a noção aceita de que solúvel em tempo polinomial é o mesmo que solúvel eficientemente na prática.

Grafo Clique O problema de reconhecimento dos grafos clique foi proposto por Roberts e Spencer em 1971 no contexto de grafos de interseção [39]. Um *conjunto completo* de um grafo $H = (V, E)$ é um subconjunto de V que induz um subgrafo completo. Uma *clique* é um conjunto completo maximal. Um grafo é um *grafo clique* se ele é o grafo de interseção dos conjuntos completos maximais de algum grafo. Foram necessários quase 40 anos para que a prova de NP-completude fosse estabelecida [1]. Não é evidente que o problema de reconhecimento pertença a NP, já que o número de completos maximais em um grafo pode ser exponencial no seu número de vértices. Considere por exemplo o grafo que consiste de um emparelhamento induzido. O grafo contém $2^{n/2}$ conjuntos independentes maximais, e portanto o seu complemento é um grafo com $2^{n/2}$ conjuntos completos maximais. A propriedade de Helly tem sido muito estudada com o objetivo de classificar a complexidade do problema de reconhecimento de grafos clique. Uma família de conjuntos $\mathcal{F} = (F_i)_{i \in I}$ é *intersectante* se a interseção de cada dois de seus membros é não vazia. A *interseção total* de \mathcal{F} é o conjunto $\bigcap \mathcal{F} = \bigcap_{i \in I} F_i$. A família \mathcal{F} tem a *propriedade de Helly*, se qualquer subfamília intersectante tem interseção total não vazia. Roberts e Spencer nesse seu artigo fundamental caracterizaram os grafos clique em termos de uma cobertura das arestas do grafo por conjuntos completos que satisfaz a propriedade de Helly, o que forneceu uma prova de que o problema de reconhecimento dos grafos clique está em NP. Szwarcfiter caracterizou os grafos clique-Helly e apresentou um algoritmo polinomial para o seu reconhecimento [44]. O problema de reconhecimento dos grafos clique foi seguidamente proposto em vários livros [7, 32, 36, 45]. Após a nossa surpreendente prova de NP-completude [1], continuamos aprofundando o estudo do problema de reconhecimento dos grafos clique para a classe dos grafos *split*. A idéia era buscar um problema que separasse em termos de complexidade as classes cordal e *split*. É conhecido que um grafo cordal tem um número linear de conjuntos completos maximais. Embora a classe dos grafos *split* seja precisamente a interseção dos grafos cordais e dos complementos de grafos cordais, são raros os problemas que separam as classes cordal e *split*, i.e., problemas que são NP-completos para cordais mas polinomiais para *split*, segundo o *The NP-completeness – an ongoing guide* de Johnson [29] e o livro de Spinrad [42]. Estabelecemos recentemente que o reconhecimento dos grafos clique *split* é NP-completo [2].

1.1.2. O Guia para Problemas Desafiadores

Na sua famosa série de colunas sobre NP-completude, Johnson [28] atualiza o seu livro seminal *The NP-completeness – an ongoing guide* [23]. Nos referimos mais precisamente à coluna sobre *graph restrictions and their effect* [29]. O objetivo é identificar problemas interessantes e classes de grafos interessantes estabelecendo o conceito de separação de complexidade. Uma classe C é dita *classe de grafos separadora* quando C separa a complexidade de dois problemas π and σ : π é NP-completo quando considerarmos como entrada para π grafos da classe C , mas σ é polinomial quando considerarmos

classe de grafos	VERTEXCOL	EDGECOL	MAXCUT
perfect	P	N	N
chordal	P	O	N
split	P	O	N
strongly chordal	P	O	O
comparability	P	N	O
bipartite	P	P	P
permutation	P	O	O
cographs	P	O	P
proper interval	P	O	O
split-proper interval	P	P	P

Tabela 1.1. N: NP-completo, P: polinomial, O: aberto

como entrada para σ grafos da classe C . Um problema π é dito *problema separador de complexidade* quando π separa duas classes de grafos $\mathcal{A} \subset \mathcal{B}$: π é polinomial quando consideramos como entrada para π grafos da classe \mathcal{A} , mas π é NP-completo quando consideramos como entrada para π grafos da classe \mathcal{B} .

A coluna sobre *graph restrictions and their effect* [29] possui uma seção dedicada aos grafos perfeitos e aos grafos de interseção. A Tabela 1.1 é uma subtabela da tabela apresentada naquela coluna onde selecionamos alguma linhas correspondentes a classes de grafos e algumas colunas correspondentes a problemas, e mantivemos a mesma notação.

É bem conhecido na área de Teoria dos Grafos que os grafos perfeitos constituem uma classe de grafos onde o problema coloração de vértices é polinomial. Por outro lado, o problema coloração de arestas é NP-completo quando restrito à classe do grafos perfeitos. Na verdade, a subclasse dos grafos de comparabilidade é uma classe de grafos separadora [27] porque separa os problemas coloração de vértices e coloração de arestas. Além disso, coloração de arestas é um problema separador de complexidade já que separa a classe dos grafos de comparabilidade da classe dos grafos bipartidos. É notável que existam poucas classes de grafos onde a complexidade de coloração de arestas é estabelecida, e é surpreendente que, desde 1985, várias classes de grafos para as quais Johnson mencionou na sua coluna como “aparentemente aberto, mas possivelmente fácil de resolver”, permanecem desafiadores, 30 anos depois. A classe dos cografos, uma classe de grafos muito restrita e estruturada definida ao proibirmos o caminho induzido por quatro vértices P_4 , admite apenas uma solução parcial [40]. A classe de grafos split-proper interval merece atenção. Ortiz, Maculan, e Szwarcfiter [34] caracterizaram os grafos que são split e proper interval, e a caracterização permitiu que coloração de arestas [34] e corte máximo [6] sejam polinomiais para esta classe. Obser-

vamos que, dois problemas clássicos: coloração de arestas e corte máximo permanecem abertos para a classe dos grafos de intervalo, uma classe de grafos de interseção muito estudada [42]. Observe que existem classes de grafos onde coloração de vértices é NP-completo e coloração de arestas é polinomial, por exemplo, grafos com um vértice universal [35]. Possivelmente mais interessante é a existência de classes de grafos onde coloração de arestas é NP-completo, mas coloração total — onde colorimos todos os elementos do grafo, vértices e arestas — é polinomial [30].

split versus cordal O fato de que as classes split e cordal concordam em relação à complexidade dos três problemas: coloração de vértices, coloração de arestas e corte máximo, sugere um padrão que tem sido investigado na literatura.

Um grafo é cordal, quando todo ciclo com pelo menos quatro vértices admite uma corda, e um grafo é split quando o seu conjunto de vértices pode ser particionado em um conjunto independente e uma clique. Os grafos split constituem uma subclasse dos grafos cordais muito estruturada, já que são exatamente os grafos tais que tanto o grafo quanto o seu complemento são ambos cordais.

Johnson, na sua coluna sobre *graph restrictions and their effect* [29], declarou que não conhecia problema que separasse as classes split e cordal do aspecto de complexidade. Spinrad, em seu livro [42] vinte anos depois, ao relatar os resultados de complexidade para classes de grafos, atualiza os problemas separadores de complexidade em relação às classes split e cordal: clique máxima e coloração de vértices são polinomiais para ambos; enquanto conjunto dominante, corte máximo e ciclo hamiltoniano são NP-completos para ambos. Existem poucos problemas separadores de complexidade em relação às classes split e cordal: empacotamento de triângulos e pathwidth, para os quais o problema é NP-completo para cordal mas polinomial para split. Spinrad [42] informa que os grafos split estão no núcleo dos algoritmos e dos resultados de dificuldade para os grafos cordais. Problemas separadores de complexidade em relação às classes split e cordal são raros. Recentemente, tentamos o problema grafo clique, mas obtivemos que o problema permanece NP-completo para grafos split [1, 2]. Possivelmente, a classe dos grafos planares fornecerá uma classe onde o problema grafo clique seja polinomial [3]. Outro problema que continua aberto é coloração de arestas, e os resultados parciais para as classes cordal, intervalo e split sugerem que o problema coloração de arestas é desafiador mesmo restrito a classes muito estruturadas [16, 34].

grafos sem ciclos com corda única Trotignon e Vušković [46] estudaram a classe C dos grafos sem ciclos com corda única. A motivação principal para investigar a classe foi encontrar um teorema que revelasse a estrutura desses grafos, um tipo de resultado que não é muito frequente na literatura. Dada uma estrutura proibida, por exemplo, ciclo

com corda única, procuramos investigar que consequência traz para a estrutura do grafo, e que problemas desafiadores tornam-se polinomiais. O teorema é do tipo: todo grafo na classe C pode ser obtido a partir de grafos básicos de C aplicando uma série de operações que “identificam” grafos de C . Outra propriedade investigada é a de família de grafos χ -limitada, introduzida por Gyárfás [25], como uma generalização natural da propriedade que define os grafos perfeitos. Uma família de grafos \mathcal{G} é χ -limitada com função χ -limitada f se, para todo subgrafo induzido G' de $G \in \mathcal{G}$, temos $\chi(G') \leq f(\omega(G'))$, onde $\chi(G')$ é o número cromático de G' e $\omega(G')$ denota o tamanho da clique máxima em G' . Os grafos perfeitos satisfazem: para todo subgrafo induzido G' de G na classe dos grafos perfeitos, temos $\chi(G') = \omega(G')$, a função χ -limitada f é a função identidade. Por outro lado, o teorema dos grafos perfeitos estabelece que um grafo G é perfeito se e somente se G e o complemento de G não admitem um ciclo ímpar sem cordas e com pelo menos 5 vértices. A pesquisa nesta área é dedicada principalmente a entender para que escolhas de subgrafos induzidos proibidos, a família resultante de grafos é χ -limitada; veja [38] para um *survey* sobre o assunto. Pelo Teorema de Vizing para coloração de arestas, a classe dos grafos de linha de grafos simples é uma classe de grafos χ -limitada com função χ -limitada $f(x) = x + 1$ (este limite superior especial é chamado o *limite de Vizing*) e, além disso, os grafos de linha são caracterizados por nove subgrafos induzidos proibidos [47]. A classe C dos grafos sem ciclos com corda única é também χ -limitada com o limite de Vizing [46]. Em [46] os seguintes resultados são obtidos para os grafos na classe C : um algoritmo $O(nm)$ para a coloração dos vértices, um algoritmo $O(n + m)$ para clique máxima, um algoritmo $O(nm)$ para o reconhecimento da classe, e a prova de NP-completude para conjunto independente máximo.

Consideramos a complexidade do problema de coloração de arestas para a classe C [31]. O problema de coloração de arestas, também conhecido como índice cromático, é o problema de determinar o menor número $\chi'(G)$ de cores necessárias para colorir as arestas do grafo G . Também investigamos subclasses obtidas da classe C quando proibimos o ciclo sem cordas com 4 vértices e quando proibimos o ciclo sem cordas com 6 vértices. As Tabelas 1.2 e 1.3 resumem os resultados obtidos, mostrando que classes com muita estrutura ainda definem problemas difíceis e desafiadores. Nas tabelas, denotamos por C a classe dos grafos sem ciclo com corda única e por Δ o grau máximo no grafo.

A classe inicialmente investigada em [31] foi a classe C dos grafos sem ciclo com corda única. Para os objetivos do trabalho, um grafo G é *básico* se G é completo, um ciclo sem cordas com pelo menos 5 vértices, um grafo fortemente 2-bipartido, ou um subgrafo induzido do grafo de Petersen ou do grafo de Heawood; e G não possui um vértice de articulação, um corte próprio com 2 vértices ou uma junção própria. Após provar que coloração de arestas é NP-completo para grafos em C , consideraremos os grafos na subclasse $C' \subset C$ que não têm C_4 . Ao proibir um ciclo sem cordas com 4 vértices

classe de grafos	$\Delta = 3$	$\Delta \geq 4$	regular
grafos em C	NP-completo	NP-completo	NP-completo
grafos em C sem C_4	NP-completo	polinomial	polinomial
grafos em C sem C_6	NP-completo	NP-completo	NP-completo
grafos em C sem C_4 , sem C_6	polinomial	polinomial	polinomial

Tabela 1.2. Dicotomia de complexidade NP-completo versus polinomial para a coloração de arestas dos grafos sem ciclo com corda única.

C_4 , evitamos a decomposição por junção própria, uma decomposição difícil para coloração de arestas [4, 40, 41], o que significa que cada grafo da classe C' que não é básico pode ser decomposto através de um vértice de articulação ou através de um corte próprio com 2 vértices. Para a classe C' , estabelecemos então uma dicotomia: coloração de arestas é NP-completo para grafos de C' com grau máximo 3 e é polinomial para grafos de C' com grau máximo *não* 3. Adicionalmente, determinamos uma condição necessária para um grafo $G \in C'$ com grau máximo 3 ser Classe 2, isto é, ter índice cromático $\chi'(G) = \Delta(G) + 1 = 4$. A condição necessária é ter um grafo P^* — subgrafo do grafo de Petersen — como um bloco básico na árvore de decomposição. Consequentemente, se proibimos ambos C_4 e C_6 , o índice cromático dos grafos sem ciclo com corda única pode ser determinado em tempo polinomial. Os resultados alcançados em [31] podem ser relacionados com várias áreas da pesquisa em coloração de arestas, como observamos a seguir.

A primeira observação diz respeito ao resultado de dicotomia encontrado para a classe class C' . A dicotomia é muito interessante porque esta é a primeira classe de grafos para a qual coloração de arestas é NP-completo para grafos com um dado grau máximo fixo Δ mas é polinomial para grafos com grau máximo $\Delta' > \Delta$. Além disso, a classe C' é a primeira classe de grafos interessante onde coloração de arestas é NP-completo em geral, mas é polinomial quando restrito a grafos regulares. É interessante observar que a classe de grafos C' é uma classe de grafos com poucos grafos regulares — somente o grafo de Petersen, o grafo de Heawood , os grafos completos e os ciclos sem cordas.

A segunda observação diz respeito à conjectura de Chetwynd e Hilton. Uma ferramenta importante para identificar classes de grafos que são Classe 2 é o conceito de *sobrecarregado* [17]. Um grafo é dito *sobrecarregado* se satisfaz $|E| > \Delta(G)|V|/2$, uma condição suficiente para que $\chi'(G) = \Delta(G) + 1$. Por exemplo, o ciclo sem cordas com 5 vértices é um grafo sobrecarregado. Uma condição suficiente mais geral é: G é *subgrafo sobrecarregado* se G possui o mesmo grau máximo que o grafo e G é sobrecarregado. Grafos que são subgrafo sobrecarregados são Classe 2 [17] e podemos verificar em tempo polinomial se um grafo é subgrafo sobrecarregado [33]. Para algumas classes

classe de grafos	$k \leq 2$	$k \geq 3$
grafos k -partidos	polinomial	NP-completo

Tabela 1.3. Dicotomia de complexidade NP-completo versus polinomial para a coloração de arestas de grafos multipartidos.

de grafos, ser subgrafo sobrecarregado é equivalente a ser Classe 2. Exemplos de tais classes são: grafos com vértice universal [35], grafos multipartidos completos [26], e grafos split com grau máximo ímpar [9]. A conjectura de Chetwynd e Hilton [10] diz que um grafo $G = (V, E)$ com $\Delta(G) > |V|/3$ é Classe 2 se e somente se é subgrafo sobrecarregado. De fato, para a maioria das classes para as quais o problema de coloração de arestas é resolvido em tempo polinomial, a equivalência “Class 2 = Subgrafo Sobrecarregado” é válida. É notável que a maioria destas classes é composta de grafos cujo grau máximo é alto — sempre superior a um terço do número de vértices. Portanto, para estas classes, a equivalência “Class 2 = Subgrafo Sobrecarregado” — e o consequente algoritmo polinomial para o problema de coloração de arestas — seriam consequência direta da Conjectura de Chetwynd e Hilton para grafos subgrafo sobrecarregados, caso a conjectura seja válida. Neste sentido, a classe C' investigada em [31, 46] apresenta um grande interesse: para grafos em C' não há limite na relação entre “número de vértices versus grau máximo”; mesmo assim, se o grau máximo não é 3, a equivalência “Class 2 = Subgrafo Sobrecarregado” é válida. Portanto, a classe dos grafos em C' com grau máximo não 3 é uma classe de grafos que não encaixa nas hipóteses da Conjectura de Chetwynd e Hilton para grafos subgrafo sobrecarregados, mas para a qual coloração de arestas é ainda solúvel em tempo polinomial através da equivalência “Class 2 = Subgrafo Sobrecarregado”.

A terceira observação é relacionada com o estudo dos *snarks* [43]. Um *snark* é um grafo cúbico sem pontes com índice cromático igual a 4. Para evitar casos triviais, snarks são em geral restritos a grafos com cintura pelo menos 5 e não contêm 3 arestas cuja remoção resulta em um grafo desconexo, com cada componente não trivial. O estudo dos snarks é relacionado fortemente ao famoso e histórico Teorema das Quatro Cores, para a coloração de mapas. Através de um resultado de [31], o único snark não trivial que *não* possui ciclo com corda única é o grafo de Petersen.

A quarta observação diz respeito ao problema de determinar o índice cromático de um grafo k -partido, isto é, um grafo cujo conjunto de vértices pode ser particionado em k conjuntos independentes. O problema de determinar o índice cromático de um grafo k -partido, já estava classificado, é polinomial para $k = 2$ [28, 29], e para grafos multipartido completos [26]. A prova de NP-completude do índice cromático dos grafos na classe C implica que coloração de arestas é NP-completo para grafos k -partidos que são r -regulares, para cada $k \geq 3$, $r \geq 3$ [31].

1.1.3. Notas Bibliográficas

Indicamos algumas publicações da autora para o leitor interessado. Para um público amplo, a autora possui um artigo de divulgação [18] na revista Ciência Hoje: “Resolver ou Verificar? Uma pergunta que vale um milhão de dólares”. Para o público especialista na área de Teoria dos Grafos e Complexidade Computacional, a autora reuniu as suas contribuições principais e apontou problemas em aberto relacionados no artigo *survey* [19]: “The P vs. NP-complete dichotomy of some challenging problems in graph theory”. A autora tem dois textos introdutórios anteriores na Jornada de Atualização em Informática (JAI), um texto sobre Emparelhamento em Grafos [20] e um texto sobre Coloração em Grafos [21].

No JAI 1999, em co-autoria com Jayme Szwarcfiter, escreveu sobre “Emparelhamento em Grafos — Algoritmos e Complexidade”. Encontrar um emparelhamento máximo em um grafo é um problema clássico no estudo de algoritmos, com muitas aplicações: designação de tarefas, determinação de rotas de veículos, problema do carteiro chinês, determinação de percursos mínimos, e muitos outros. O artigo histórico de Edmonds que descreveu um algoritmo $O(n^4)$ para o problema geral de emparelhamento, na verdade, introduziu a noção de algoritmo de tempo polinomial. Implementações mais eficientes do algoritmo de Edmonds foram apresentadas por vários pesquisadores como Lawler (algoritmo $O(n^3)$), Hopcroft e Karp (algoritmo $O(m \sqrt{n})$ para o caso bipartido), Micali e Vazirani (algoritmo $O(m \sqrt{n})$ para o caso geral, o mais eficiente até o momento). O texto apresenta uma introdução ao problema de emparelhamento em grafos, e um estudo de seus algoritmos e complexidade. Apresentamos algoritmos eficientes para os quatro problemas relacionados com encontrar um emparelhamento de cardinalidade máxima ou de peso máximo, em grafos bipartidos ou em grafos quaisquer. Estes quatro problemas são todos casos particulares do problema de emparelhamento de peso máximo em grafos quaisquer, mas é interessante considerá-los em ordem crescente de dificuldade por razões de ordem didática [20].

No JAI 1997, em co-autoria com João Meidanis e Célia Mello, escreveu sobre “Coloração em Grafos”. O texto é uma introdução à coloração em grafos. Consideramos dois tipos de problemas de coloração em grafos: coloração de vértices e coloração de arestas. Aplicamos a estes problemas técnicas clássicas de coloração tais como: algoritmos gulosos, decomposição e alteração estrutural. Consideramos classes de grafos para as quais tais técnicas resolvem estes problemas de coloração eficientemente. O estudo de coloração, um tópico básico em Teoria dos Grafos, surgiu a partir do conhecido “problema das quatro cores”. Em 1852, Francis Guthrie questionou se todo grafo planar poderia ser colorido com quatro cores. Embora estivesse claro que quatro cores eram necessárias, a questão se referia ao número mínimo de cores, i.e., quantas cores são suficientes para colorir qualquer grafo planar. Esta questão foi respondida afirmativamente por Appel e Haken, usando computador, após mais de 100 anos. Robertson,

Sanders, Seymour e Thomas mais recentemente confirmaram esta resposta fornecendo uma prova mais simples, mais aceita pela comunidade matemática, embora ainda use o computador. O problema das quatro cores corresponde à coloração dos vértices de um grafo. Mais precisamente, corresponde à coloração mínima dos vértices de um grafo. Outros problemas de coloração existem, tais como coloração das arestas, coloração total (vértices e arestas), coloração a partir de conjuntos de cores atribuídos préviamente aos vértices ou às arestas de um grafo. O escopo do texto, no entanto, é mais restrito: coloração mínima de vértices e coloração mínima de arestas em classes de grafos [21].

Referências

- [1] L. Alcon, L. Faria, C. M. H. de Figueiredo, and M. Gutierrez, The complexity of clique graph recognition, *Theoret. Comput. Sci.* **410** (2009) 2072–2083.
- [2] L. Alcon, L. Faria, C. M. H. de Figueiredo, and M. Gutierrez, Split clique graph complexity, *Theoret. Comput. Sci.* **506** (2013) 29–42.
- [3] L. Alcón, and M. Gutierrez, Cliques and extended triangles. A necessary condition for planar clique graphs, *Discrete Appl. Math.* **141** (2004) 3–17.
- [4] M. M. Barbosa, C. P. de Mello, and J. Meidanis, Local conditions for edge-colouring of cographs, *Congr. Numer.* **133** (1998) 45–55.
- [5] C. Berge, and V. Chvátal, *Topics on Perfect Graphs*, North-Holland Mathematics Studies, 88. *Annals of Discrete Mathematics*, 21. North-Holland Publishing Co., Amsterdam, 1984.
- [6] H. L. Bodlaender, C. M. H. de Figueiredo, M. Gutierrez, T. Kloks, and R. Niedermeier, Simple max-cut for split-indifference graphs and graphs with few P_4 's, Proc. of Third International Workshop on Experimental and Efficient Algorithms (WEA 2004). *Lecture Notes in Comput. Sci.* 3059 (2004), 87–99.
- [7] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.
- [8] K. Cameron, *Polyhedral and Algorithmic Ramifications of Antichains*, Ph.D. Thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, 1982.
- [9] B. L. Chen, H.-L. Fu, and M. T. Ko, Total chromatic number and chromatic index of split graphs, *J. Combin. Math. Combin. Comput.* **17** (1995) 137–146.
- [10] A. G. Chetwynd, and A. J. W. Hilton, Star multigraphs with three vertices of maximum degree, *Math. Proc. Cambridge Philos. Soc.* **100** (1986) 303–317.

- [11] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vušković, Recognizing Berge graphs, *Combinatorica* **25** (2005) 143–186.
- [12] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, The strong perfect graph theorem, *Ann. of Math.* **164** (2006) 51–229.
- [13] V. Chvátal, Star-cutsets and perfect graphs, *J. Combin. Theory Ser. B* **39** (1985) 189–199.
- [14] T. Feder, P. Hell, S. Klein, and R. Motwani, List partitions, *SIAM J. Discrete Math.* **16** (2003) 449–478.
- [15] C. M. H. de Figueiredo, S. Klein, Y. Kohayakawa, and B.A. Reed, Finding skew partitions efficiently, *J. Algorithms* **37** (2000) 505–521.
- [16] C. M. H. de Figueiredo, J. Meidanis, and C. P. de Mello, On edge-colouring indifference graphs, *Theoret. Comput. Sci.* **181** (1997) 91–106.
- [17] C. M. H. de Figueiredo, J. Meidanis, and C. P. de Mello, Local conditions for edge-coloring, *J. Combin. Math. Combin. Comput.* **32** (2000) 79–91.
- [18] C. M. H. de Figueiredo, Resolver ou Verificar? Uma pergunta que vale um milhão de dólares. *Ciência Hoje*, Rio de Janeiro pp. 42–46, novembro 2011.
- [19] C. M. H. de Figueiredo, The P vs. NP-complete dichotomy of some challenging problems in graph theory, *Discrete Appl. Math.* **160** (2012) 2681–2693.
- [20] C. M. H. de Figueiredo, and J. L. Szwarcfiter, Emparelhamentos em Grafos: Algoritmos e Complexidade, *XIX Congresso da Sociedade Brasileira de Computação. (Org.). XVIII Jornada de Atualização em Informática*. 1999, pp. 127–161.
- [21] C. M. H. de Figueiredo, J. Meidanis, and C. P. Mello, Coloração em Grafos, *XVII Congresso da Sociedade Brasileira de Computação. (Org.). XVI Jornada de Atualização em Informática* 1997, pp. 01–45.
- [22] L. Fortnow, The status of the P versus NP problem, *Communications of the ACM* **52** (2009) 78–86.
- [23] M. R. Garey, and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-completeness*. WH Freeman, New York, 1979.
- [24] M. Grötschel, L. Lovász, and A. Schrijver, Polynomial Algorithms for Perfect Graphs, In *Topics on Perfect Graphs*, C. Berge and V. Chvátal, eds., North-Holland Mathematics Studies 88, Annals of Discrete Mathematics 21, North-Holland Publishing Co., Amsterdam, 1984, pp. 325–356.

- [25] A. Gyárfás, Problems from the world surrounding perfect graphs, *Zastos. Mat.* **19** (1987) 413–441.
- [26] D. G. Hoffman, and C. A. Rodger, The chromatic index of complete multipartite graphs, *J. Graph Theory* **16** (1992) 159–163.
- [27] I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Comput.* **10** (1981) 718–720.
- [28] D. S. Johnson, *NP-Completeness Columns*, available at <http://www2.research.att.com/~dsj/columns/>
- [29] D. S. Johnson, Graph restrictions and their effect, *J. Algorithms* **6** (1985) 434–451.
- [30] R. C. S. Machado, and C. M. H. de Figueiredo, Complexity separating classes for edge-colouring and total-colouring, *J. Braz. Comput. Soc.* **17** (2011) 281–285.
- [31] R. C. S. Machado, C. M. H. de Figueiredo, and K. Vušković, Chromatic index of graphs with no cycle with a unique chord, *Theoret. Comput. Sci.* **411** (2010) 1221–1234.
- [32] T. A. McKee, and F. R. McMorris, *Topics in Intersection Graph Theory*, SIAM Monographs on Discrete Mathematics and Applications, 1999.
- [33] T. Niessen, How to find overfull subgraphs in graphs with large maximum degree, *Electron. J. Combin.* **8** (2001) #R7.
- [34] C. Ortiz Z., N. Maculan, and J. L. Szwarcfiter, Characterizing and edge-colouring split-indifference graphs, *Discrete Appl. Math.* **82** (1998) 209–217.
- [35] M. Plantholt, The chromatic index of graphs with a spanning star, *J. Graph Theory* **5** (1981) 45–53.
- [36] E. Prisner, *Graph Dynamics*, Pitman Research Notes in Mathematics 338, Longman, 1995.
- [37] J. L. Ramirez Alfonsin, and B. A. Reed, *Perfect graphs*, Wiley-Interscience Series in Discrete Mathematics and Optimization, 2001.
- [38] B. Randerath, and I. Schiermeyer. Vertex colouring and forbidden subgraphs—a survey, *Graphs Combin.* **20** (2004) 1–40.
- [39] F. Roberts, and J. Spencer, A characterization of clique graphs, *J. Combin. Theory Ser. B* **10** (1971) 102–108.

- [40] C. Simone, and C. P. de Mello, Edge-colouring of join graphs, *Theoret. Comput. Sci.* **355** (2006) 364–370.
- [41] C. Simone, and A. Galluccio, Edge-colouring of regular graphs of large degree, *Theoret. Comput. Sci.* **389** (2007) 91–99.
- [42] J. P. Spinrad, *Efficient graph representations*, Fields Institute Monographs, 19. AMS, 2003.
- [43] E. Steffen, Classifications and characterizations of snarks, *Discrete Math.* **188** (1998) 183–203.
- [44] J. L. Szwarcfiter, Recognizing clique-Helly graphs, *Ars Combin.* **25** (1997) 29–32.
- [45] J. L. Szwarcfiter, A Survey on Clique Graphs, In *Recent Advances in Algorithms and Combinatorics*, C. Linhares-Sales and B. Reed, eds., CMS Books Math./Ouvrages Math. SMC, 11, Springer, New York, 2003, pp. 109–136.
- [46] N. Trotignon, and K. Vušković, A structure theorem for graphs with no cycle with a unique chord and its consequences, *J. Graph Theory* **63** (2010) 31–67.
- [47] D. B. West, *Introduction to Graph Theory*. Second edition. Prentice Hall, 2001.

1.2. Lógica Para Computação: Uma Introdução Curta - Luís C. Lamb

Abstract

The understanding of logical systems is fundamental to Computer Science research. Even the notion of computability has originated in logic-based research. Several logics have been used and developed in Computing since the dawn of this science, in particular in the second half of the XX Century. The use of logical methods has been fundamental to several fields in Computing research. These include, e.g. the specification and verification of computational systems, database systems design, descriptive complexity, integrated circuits design, semantic web, artificial intelligence and distributed systems. This chapter presents a brief introduction to computational logic. We shall also present historical notions on the evolution of logic in Computer Science, illustrating its use by means of applications in relevant fields.

Resumo

O entendimento de sistemas lógicos é fundamental em Ciência da Computação. A própria noção de computabilidade é originária de estudos em lógica. Diversas lógicas têm sido utilizadas e desenvolvidas em Computação desde o estabelecimento desta como ciência organizada, particularmente a partir da segunda metade do Século XX. O uso de métodos lógicos tem sido fundamentais em múltiplas subáreas da computação. Estas áreas incluem: especificação e verificação de sistemas computacionais, projeto de sistemas de bancos de dados, projeto lógico de circuitos integrados, complexidade descritiva, web semântica, inteligência artificial e sistemas distribuídos. Este capítulo apresenta uma breve introdução à lógica computacional. Apresentaremos, também, noções históricas sobre a evolução da lógica em Ciência da Computação, ilustrando com algumas aplicações em áreas relevantes.

1.2.1. Introdução: Brevíssimas Considerações Históricas

It takes an extraordinary intelligence to contemplate the obvious.

Alfred North Whitehead

Lógica é uma área organizada do domínio do conhecimento humano. É uma área de estudo caracterizada por conceitos e definições precisas. O estudo das técnicas e métodos de lógica(s) permite a melhor definição, entendimento e raciocínio sobre diversos domínios do conhecimento.¹

Na Grécia antiga (“clássica”), os *Estoicos*² estudaram e definiram noções iniciais de lógica proposicional e, preliminarmente, de inferência. O nome de Aristóteles, fundamental na história da ciência define os silogismos (padrões de raciocínio em que premissas tem uma conclusão lógica) e o raciocínio com quantificadores (séc. IV A.C.). Exemplo simples de silogismo: *Todos os lógicos são filósofos; Luís é lógico; Portanto, Luís é filósofo;* silogismos foram utilizados até a idade média, servindo, inclusive para a formalização de textos escolásticos. No período medieval, alguns lógicos se destacam, como Pierre Abélard (França), no Século XIII, e William of Ockham (Inglaterra) no Século XIV. Ockham é referenciado até hoje pelo conceito da “navalha de Ockham”, mas também introduziu princípios similares ao que conhecemos hoje como Leis de *De Morgan* em linguagem natural, além de lógicas tri-valoradas. Abélard publica o *Logica ingredientibus*, em 1121 (lógica para principiantes) e destaca-se como grande propagador do sistema de Aristóteles. Ambos fazem uso do sistema silogístico para verificação de argumentos. No mesmo período, Ramon Llull (*latim: Lulio*), no Séc. XIII, desenvolve estudos para verificar automaticamente a verdade de silogismos, desenvolvendo um sistema de raciocínio como computação. É importante ressaltar que o trabalho de Llull, de certa forma, influenciou o trabalho de Leibniz.

Durante um longo período, o conhecimento acadêmico foi pouco desenvolvido ou relatado e reportado, algumas vezes justificado pelo período da idade média no mundo ocidental. Na idade média, o ensino da lógica era parte do *trivium*: conjunto

¹O objetivo deste texto não é oferecer uma história completa da lógica. As considerações históricas são apresentadas, apenas, para estimular o leitor a buscar informações adicionais sobre a evolução da lógica e seu impacto na ciência da computação. O texto também *não oferece* um curso completo de lógica; apenas introduz, de forma não exaustiva, conceitos e noções básicas e - em reduzida síntese - sua relação com a ciência da computação. O leitor é convidado a consultar as referências bibliográficas citadas no texto.

²O pensamento estoico tem como pioneiro Zeno (ou Zenão) de Cílio (344-262 AC). O pensamento estoico foi predominante na filosofia ocidental até o crescimento da doutrina cristã. Para os estoicos, a lógica inclui a análise de argumentos, a retórica, gramática, teoria de conceitos, percepções, proposições, pensamentos, o que - de certa forma - poderia ser visto como epistemologia ou filosofia da linguagem. No entanto, não há espaço suficiente neste capítulo para maiores detalhes sobre o estoicismo. Uma referência na moderna “Stanford Encyclopedia of Philosophy” [1] é uma boa bibliografia inicial sobre o assunto.

de disciplinas básicas das artes liberais: gramática, lógica e retórica. O trivium era base para o *quadrivium*, conjunto de disciplinas formado pela aritmética, geometria, música e astronomia. Neste período, há desenvolvimentos relevantes no oriente médio e vizinhança, particularmente na Pérsia, sub-continento asiático (Índia) e Oriente.³ Outras contribuições significativas são de Gottfried Willhem Leibniz (1646-1716). Leibniz, grande nome da matemática, idealizou uma linguagem universal a *lingua characteristica universalis* para expressar todo conhecimento humano. Desenvolveu também o *Calculus ratiocinator*: modelo universal teórico de cálculo lógico - executado por máquinas - para derivar relações lógicas. Esta contribuição é relevante devido ao objetivo computacional. Tanto que Leibniz é referido como o “Patron saint of Computer Science” por Moshe Vardi, na *Communications of the ACM*, no Editorial de Dezembro de 2011.



Figura 1.1. Aristoteles, G. W. Leibniz

No século XIX há grandes desenvolvimentos em lógica matemática. George Boole (1815-1864) nome de forte impacto em matemática (posteriormente em ciência da computação, economia e engenharia) desenvolve a lógica proposicional, sob rigorosa formalização. Historiadores da ciência afirmam que Boole, de fato, introduz a lógica simbólica em *An Investigation of the Laws of Thought* (1854). No mesmo século, em torno de 1879, Gottlob Frege (1848-1925), no *Begriffsschrift*, formaliza a Lógica de Primeira Ordem, introduzindo definições precisas da quantificação existencial e universal em lógica matemática. Augustus De Morgan (1806-1871) formaliza a lógica das relações e define *indução matemática* de forma rigorosa - técnica hoje muito utilizada em ciência da computação - além de formalizar as leis que levam seu nome. Além desses, Charles Sanders Peirce (1839-1914) define também o raciocínio abdutivo, propõe uma definição rigorosa de indução matemática e princípios de lógica relacional, estendendo trabalho de De Morgan. Ademais, mostra que a utilização do operador *nor* permite representar todas as funções booleanas. No final do século XIX, Bertrand Russell mostra que o sistema de Frege é inconsistente. Isto leva Russell ao desenvolvimento da teoria de tipos em lógica matemática (que terá certa influência e inspiração sobre a teoria de ti-

³Para melhor entendimento dos estudos em lógica desenvolvidos neste período histórico recomendamos a consulta do *Handbook of Philosophical Logic* e *Handbook of the History of Logic*. Estas importantes obras modernas de organização do conhecimento sobre lógica pura e aplicada publicaram capítulos específicos sobre o desenvolvimento da lógica nestas regiões e nestes períodos históricos [20, 22].

pos em ciência da computação, particularmente na área de linguagens de programação).



Figura 1.2. Grandes Lógicos do Século XIX: Boole, De Morgan, Frege, Peirce

Entre 1903 e 1910, Alfred North Whitehead (1861-1947) e Bertrand Russell (1872-1970) escrevem o monumental *Principia Mathematica* [52], em 3 volumes, com mais de 2 mil páginas. O objetivo era formalizar na linguagem da lógica, utilizando axiomas e regras de inferência, todo o conhecimento matemático. A partir deste período, a lógica matemática e a teoria de conjuntos estabelecem-se como fundamentos da matemática moderna.⁴

Luitzen Egbertus Jan Brouwer (1881-1966), desenvolve a lógica intuicionista [39, 49]. A lógica intuicionista (ou construtiva) rejeita a lei do meio excluído em seu sistema, a eliminação da dupla negação $\neg\neg\alpha \not\vdash \alpha$, assim como provas por contradição. Assim, as provas nesta lógica são construtivas, o que remete à relação com conceitos de computação. O isomorfismo de Curry-Howard, simplificadamente, relaciona provas construtivas (intuicionistas) e programas. A lógica intuicionista tem sido pesquisada

⁴O *Principia* ou PM, como também é conhecido, foi considerado um dos 100 livros de não-ficção mais importantes do Século XX pelo New York Times(<http://www.nytimes.com/library/books/042999best-nonfiction-list.html>). Whitehead e Russell foram filósofos de grande influência no Século XX. Russell foi considerado por muitos especialistas o “filósofo do século” [42]. Whitehead foi considerado genial pela famosa intelectual da primeira metade do século, Gertrude Stein, que nas primeiras frases da “Autobiografia de Alice B. Toklas” (na verdade sua própria autobiografia, uma obra vanguardista na cultura ocidental) afirmou “... that only three times in my life have I met a genius ... the three geniuses of whom I wish to speak are Gertrude Stein, Pablo Picasso and Alfred Whitehead.”



Figura 1.3. Alfred Whitehead, Bertrand Russell, David Hilbert, L.E.J. Brouwer, Alfred Tarski

como um sistema lógico que pode ser fundamental em diversas áreas da ciência da computação, incluindo na teoria de tipos de linguagens de programação.

Na década de 1920, David Hilbert (1862-1943), um dos matemáticos mais influentes do início do século XX, propôs a busca por uma nova fundamentação para a matemática. Sinteticamente, Hilbert queria mostrar (1) a completude da matemática: todas as verdades podem ser provadas. (2) a consistência da matemática: uma afirmação matemática e sua contradição não podem ser ambas provadas através de um sistema lógico. (3) a decidibilidade: existe uma forma mecânica de mostrar se uma afirmação matemática arbitrária qualquer é verdadeira ou falsa. Através de um procedimento formal, Hilbert queria mostrar que era possível construir todas as verdades matemáticas. Deste programa deriva-se o *Entscheidungsproblem* (ou “problema da decisão”) - demonstrar a (in)existência de um algoritmo para determinar se um enunciado da lógica de primeira ordem pode ser provado. Em 1922, Ludwig Wittgenstein publica o *Tractatus Logico-Philosophicus* [53]. O *Tractatus* é uma obra influente na filosofia analítica e estabeleceu o seu autor como um dos filósofos mais importantes do século XX. Nesta obra, Wittgenstein define, entre diversas conceituações notáveis, a noção de tabelas-verdade, na forma em que passam a ser utilizadas até hoje.

Na década de 1930 Kurt Gödel, Alan Turing e Alonzo Church estudam as limitações da lógica de primeira ordem.⁵ Entre as contribuições fundamentais de Gödel, citamos o seu primeiro teorema da incompletude: um sistema axiomático suficientemente poderoso para descrever a aritmética dos naturais, não é completo e o segundo teorema de Gödel: em um sistema axiomático suficientemente poderoso para descrever a aritmética dos naturais, a consistência destes axiomas não pode ser provada pelo próprio sistema. Tais resultados responderam negativamente ao programa de Hilbert, mostrando não ser possível definir um sistema axiomático correto e completo para toda matemática. É interessante observar que estudos em decidibilidade contribuem decisivamente para o desenvolvimento da noção de computabilidade. Neste mesmo período histórico e até os anos 1950, a definição precisa de consequência lógica, a noção de verdade em uma estrutura e trabalhos fundamentais na teoria de modelos são desenvolvidos por Alfred Tarski.

Além disso, Gödel desenvolve um sistema completo e correto para a lógica de primeira ordem. Church encontra uma solução negativa ao *Entscheidungsproblem* [5]: é impossível decidir algorítmicamente se afirmações da aritmética são verdadeiras ou falsas; Church define a noção de *calculabilidade* e também introduz o cálculo- λ para estudo das funções referidas por ele como calculáveis - mas referidas por Turing como *computáveis*. Posteriormente, o cálculo- λ também passa a ter forte influência na ciência da computação, principalmente na área de linguagens de programação, a partir da dé-

⁵A contribuição de Kurt Gödel foi inclusive popularizada pelos grandes meios de comunicação da imprensa. Em 1999, Gödel foi escolhido o matemático do século pela revista *Time Magazine*.

cada de 1960. De grande influência na comunidade de lógica, Church orientou as teses de nomes relevantes em lógica e ciência da computação como: A.M. Turing, S. Kleene, J.B. Rosser, D.S. Scott, M.O. Rabin, H. Rogers Jr., R. Smullyan e Leon Henkin.

Atualmente, em poucos textos a referência ao termo *calculabilidade* é encontrada. No famoso artigo de Turing [46], ele faz referência pioneira ao termo “computable” enquanto esclarece que o conceito de “calculable” (calculável) de Church ao estudar funções calculáveis é equivalente à sua definição de computável [46]:

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In §8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel [1]. These results have valuable applications. In particular, it is shown (§11) that the Hilbertian Entscheidungsproblem can have no solution. In a recent paper Alonzo Church [2] has introduced an idea of “effective calculability”, which is equivalent to my “computability”, but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem [3]. The proof of equivalence between “computability” and “effective calculability” is outlined in an appendix to the present paper.

- [1] Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38(1931):173-198.
- [2] An unsolvable problem of elementary number theory, *Amer. J. Math.*, 58(1936): 345-363.
- [3] A note on the Entscheidungsproblem. *J. of Symbolic Logic*, 1(1936): 40-41.

Ademais, em [46] as noções de algoritmo, procedimento efetivo e computabilidade são explicitadas. A partir das contribuições de Turing, Church, Gödel e muitos outros, a Ciência da Computação passa a ser desenvolvida sob rigor científico e, fundamentalmente, lógico-matemático. As contribuições de Church e Turing levam à formulação da Tese de Church-Turing: uma função sobre números inteiros é computável se, e somente se, ela é computável em uma Máquina de Turing. Outra ideia de Turing - o programa armazenado em fita tem grande impacto, inclusive tecnológico. Este princípio muito simples influenciou von Neumann no desenvolvimento do conceito de programa armazenado em memória.

A partir do final da segunda guerra mundial, o desenvolvimento da Ciência da Computação acontece de forma muito acelerada. O próprio desenvolvimento do transistor, no final da década de 1940, a construção dos primeiros computadores e o desenvolvimento do modelo de arquitetura de computadores conhecido como modelo de von Neumann, levam a impressionantes impactos sociais e econômicos. Ainda nesta

época, Turing novamente causa impacto, ao levantar questões relacionados ao raciocínio e aprendizado de máquinas, i.e., pavimentando o caminho para o surgimento da Inteligência Artificial. Evidentemente, para formalizar raciocínio torna-se necessário o desenvolvimento de novos sistemas lógicos, “implementáveis” em computadores. A partir da década de 1960 novas lógicas são desenvolvidas em ciência da computação, tendo em vista a evolução da ciência e a necessidade de especificar, verificar e raciocinar sobre programas, sistemas e construções computacionais. Em suma, o impacto da lógica em computação foi tão significativo que muitos pesquisadores se referem à lógica como o “cálculo da computação” [25, 36].



Figura 1.4. Alan Turing, FRS (1912-54), Kurt Gödel (1906-78), Alonzo Church (1903-95)

1.2.2. Lógica em Computação: Uma Breve Introdução

A seguir, apresentamos uma introdução muito básica à lógica proposicional. Há excelentes livros que recomendamos para o leitor, com extenso material sobre o assunto, inclusive sobre aspectos relacionados à Ciência da Computação. Um dos melhores textos em língua portuguesa é o livro de Corrêa da Silva, Finger e Melo [7]. Neste livro, são abordados em detalhe temas como SAT (satisfatibilidade), prova automática de teoremas e verificação de programas. Este capítulo tem como objetivo, apenas, oferecer uma breve introdução à lógica em computação. Na literatura há outras obras de grande abrangência e qualidade sobre lógica em computação, e.g. [3, 19, 29].

1.2.3. Lógica Proposicional

Lembre-se que para falar sobre algum domínio, é necessário estabelecer (isto é, definir) uma *linguagem*. Em sistemas lógicos, as noções de sintaxe, semântica, e teoria de prova (sistemas de prova) são fundamentais. A teoria de prova se refere à forma sintática de obter ou identificar as afirmações válidas da linguagem.

Também dizemos que a sintaxe se refere às regras utilizadas para construir as fórmulas de um sistema lógico. Por sua vez, a semântica se refere ao significado destas fórmulas bem-formadas (formadas de acordo com regras definidas de sintaxe), mapeando as sentenças a uma interpretação. Em ciência da computação, estes conceitos também são essenciais, tanto do ponto de vista fundamental, quanto do ponto de vista de aplicações, como em linguagens de programação. Assim, o conhecimento básico de

lógica clássica torna-se útil em um grande número de aplicações. Tipicamente, a lógica clássica inclui e tem como objeto de estudo a lógica proposicional e a lógica de predicados de primeira ordem. Iniciaremos, desta forma, nossa abordagem através de breves introduções à lógica proposicional e à lógica de predicados.

Em lógica proposicional as afirmações atômicas são *proposições*: fatos que podem ser verdadeiros/falsos em um determinado mundo possível/situação. Para representar variáveis proposicionais, por exemplo, podemos usar $p, q, r\dots$; para representar os conetivos (ditos “Booleanos”) tipicamente utilizamos $\neg, \vee, \wedge, \rightarrow$ (ou \supset)⁶. Estes símbolos de conetivos representam negação (conetivo unário), disjunção (“ou” lógico), conjunção (“e” lógico) e implicação material (“se então”, informalmente). Exemplos de formulae podem ser construídos a partir destes símbolos da linguagem proposicional: $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$; $(p \rightarrow q) \rightarrow (\neg p \vee q)$; $(p \rightarrow q) \rightarrow p$.

Note que a lógica proposicional tem expressividade limitada. Por exemplo, a afirmação *todos os homens são mortais*, é expresso como um átomo proposicional, abstraindo-se relações e quantificações. Os conetivos “Booleanos” têm uma interpretação similar à utilizada em linguagem natural. Assim, $p \wedge q$ intuitivamente significa que p e q são interpretados como verdadeiros em uma determinada situação. Lógicas são utilizadas para aplicações, por exemplo, em representação do conhecimento (humano). Portanto, encontrar e estabelecer correspondência entre lógica(s) e linguagem natural é essencial nestas aplicações. Esta “tradução” nem sempre é simples. Lembre que a lógica clássica nos permite representar somente afirmações. A idéia fundamental na tradução é tentar extrair proposições atômicas de uma afirmação e combiná-las através de conetivos. Por exemplo, “Maria gosta de João e de Pedro” pode ser “traduzida” como “Maria gosta de João \wedge Maria gosta de Pedro” e formalizada como $(p \wedge q)$. Por sua vez, a afirmação (proposição) “Spinoza não gosta de Tomás” pode ser representada como $\neg(\text{Spinoza gosta de Tomás})$ e formalizada simplesmente como $\neg p$. Note que esta formalização proposicional abstrai relações entre objetos/pessoas das afirmações em linguagem natural. Finalmente, um exemplo envolvendo implicação material (ou um “condicional”): “Se o Internacional vence, então todos estamos felizes” pode ser traduzida como “Inter vencedor \rightarrow felizes” e formalizada $p \supset q$ ou $p \rightarrow q$. No entanto, como a linguagem natural é ambígua, encontrar uma tradução aceita universalmente nem sempre é possível: por exemplo, “Pedro será rico e famoso” não pode ser apenas

⁶A sintaxe para o condicional $p \rightarrow q$ é hoje a mais comumente utilizada. Na notação do Principia Mathematica [52] a implicação material seria representada como $p \supset q$. No entanto, deve-se observar que existem diversos condicionais em lógica(s). O condicional da lógica clássica é conhecido como “implicação material”, sendo que $p \rightarrow q$ é logicamente equivalente a $\neg p \vee q$. No entanto, esta equivalência não é válida em outros sistemas lógicos não-clássicos, por exemplo em lógica intuicionista, na qual esta equivalência não é um teorema. Há diversas outras formalizações de condicionais: contrafatuais, condicionais estritos, causais e outros. No entanto, o escopo deste capítulo é restrito, e não exploraremos estes “condicionais não-clássicos” [4].

traduzido como “Pedro será rico \wedge Pedro será famoso”, pois Pedro poderá ficar famoso somente após perder sua fortuna em um cassino, o que possivelmente seria melhor formalizado com uma *lógica temporal*. Algumas fórmulas são relativamente simples: Por exemplo, “está chovendo \rightarrow Candide não sai de casa” pode ser traduzido como “se está chovendo então Candide não sai de casa”. Entretanto, *não* devemos ler a fórmula $p \rightarrow q$ como “p causa q”, pois a relação de causa e efeito tem interpretação distinta da implicação material da lógica clássica.

É claro que nem sempre existe uma correspondência “trivial” entre aquilo que é expresso através de uma linguagem natural e através de uma linguagem simbólica de uma lógica. Em computação, a fim de expressarmos as afirmações e o conhecimento sobre um determinado domínio, idealmente imaginamos que não existam ambiguidades nas interpretações de fórmulas. Na lógica clássica, trabalhamos com valores-verdade de fórmulas em situações nas quais determina-se se os componentes (átomos) de fórmulas são verdadeiras ou não. Se pensarmos num programa de computador, avaliaríamos um teste condicional (um “if”) em função dos valores das variáveis do programa. Para átomos p, q, r, \dots uma situação especificará os valores-verdade dos mesmos, i.e. se eles são verdadeiros ou falsos nesta situação. Note que os valores-verdade dos átomos podem ser diferentes em diferentes situações. O valor-verdade de uma fórmula proposicional em uma situação é definida como a seguir. É conveniente incluir \top e \perp como as fórmulas que são sempre verdadeiras e sempre falsas, respectivamente. Assim, \top e \perp denotam, também, tautologias e contradições, respectivamente. \top é verdadeiro e \perp é falso; $\neg p$ verdadeiro se p é falso, e falso se p é verdadeiro. $p \wedge q$ é verdadeiro se p, q são ambos verdadeiros; caso contrário, $p \wedge q$ é falso. $p \vee q$ é verdadeiro se um ou mais dentre p, q são verdadeiros, falso em caso contrário. $p \rightarrow q$ é verdadeiro se p é falso ou q é verdadeiro (ou ambos). Caso contrário, a implicação é falsa.⁷ As noções de *verdade*, *validade* e *equivalência* são essenciais em lógica. Uma fórmula proposicional é logicamente válida se ela é verdadeira em qualquer situação. Escreve-se $\models p$ se p é válida, para uma fórmula arbitrária p . Fórmulas proposicionais válidas são também conhecidas como *tautologias*.

Por sua vez, uma fórmula p é uma *contradição* se ela for falsa em todas interpretações. Uma fórmula proposicional é *satisfável* (ou ainda *consistente*) se ela é verdadeira em pelo menos uma situação. Duas fórmulas p, q são logicamente equivalentes se elas são verdadeiras em exatamente nas mesmas situações. Podemos escrever $p \equiv q$ neste caso. Outra noção relevante é a de consistência de um conjunto de fórmulas. Um conjunto de fórmulas bem formadas $\{A_1, A_2, \dots, A_n\}$ é consistente se existe uma

⁷Podemos expressar o significado definido acima através de *tabelas-verdade*. Ludwig Wittgenstein, no *Tractatus Logico-Philosophicus* [53] define a noção de tabelas-verdade, na forma em que passaram a ser utilizadas até hoje. Cada linha de uma tabela-verdade corresponderá a uma *interpretação* da fórmula para a qual construímos a tabela.

interpretação de sua conjunção que é verdadeira; i.e. se valores podem ser atribuídos a todas as sentenças atômicas no conjunto de fórmulas $\{A_1, A_2, \dots, A_n\}$ que fazem com que as fórmulas sejam verdadeiras. Por exemplo, o conjunto $\{p \rightarrow q, \neg p, \neg q\}$ é consistente, enquanto que o conjunto $\{r \rightarrow s, r, \neg s\}$ não é consistente.

Dizemos que uma fórmula q é *consequência lógica* de um conjunto de fórmulas p_1, p_2, \dots, p_n , representado por $p_1, p_2, \dots, p_n \models q$ se em toda situação (interpretação) na qual p_1, p_2, \dots, p_n são todas fórmulas verdadeiras, q também é verdadeira. Note que esta é uma noção de consequência *semântica*. Podemos, também definir consequência lógica como uma noção *sintática*; neste caso, $p_1, p_2, \dots, p_n \vdash q$ significa que existe um *prova* de q a partir das fórmulas p_1, p_2, \dots, p_n , utilizando um sistema de prova (e.g. sistema de dedução natural, tableaux, etc). Uma *prova* é uma sequência (finita) de fórmulas bem-formadas que são deduzidas a partir de outras fórmulas através do uso de regras de um sistema de provas ou de axiomas. Resultados anteriormente já provados (teoremas) podem ser utilizados como *lemas* em provas. Note que um sistema de provas utiliza regras puramente sintáticas. Podemos dizer que q pode ser *provado* a partir das premissas p_1, p_2, \dots, p_n . Assim, as noções de consequência lógica no nível sintático ($p_1, p_2, \dots, p_n \vdash q$) e semântico ($p_1, p_2, \dots, p_n \models q$) são claramente distintas. O conjunto de fórmulas (premissas) p_1, p_2, \dots, p_n é também chamado de teoria. A construção sintática $p_1, p_2, \dots, p_n \vdash q$ é denominada, também, de *sequente*. As fórmulas à esquerda de \vdash são chamadas de antecedente, enquanto que a fórmula à direita de \vdash é chamada de consequente. Por sua vez, *teoremas* são fórmulas que seguem logicamente de outras fórmulas válidas; são afirmações deduzidas a partir de outras afirmações (e que são, obviamente, válidas) em um cálculo formal. Formalmente, no entanto, um teorema é a última linha de uma prova. A notação $\vdash \alpha$ denota que α é um teorema. Existem diversos métodos para verificação de validade de fórmulas, incluindo: (1) tabelas-verdade, onde simplesmente verificamos mecanicamente todas as situações relevantes; têm crescimento exponencial no número de subfórmulas atômicas e são aplicáveis somente à lógica proposicional; (2) argumentação direta (utilizado por matemáticos/lógicos), que é relativamente rápida, desde que o usuário seja familiarizado com o método; (3) Sistemas de prova: sistemas de Hilbert (axiomáticos), tableaux, dedução natural e métodos automáticos (provadores automáticos de teoremas - “automated theorem provers”).

Propriedades importantes de sistemas lógicos (ditas metapropriedades) incluem a *correção* e a *completude*. Se todas as fórmulas válidas de um sistema lógico são provadas por um sistema de provas para este sistema, ele é dito *completo* em relação ao modelo semântico; enquanto que a *correção* garante que as provas obtidas por este sistema de prova efetivamente são válidas em relação ao modelo semântico. Por exemplo, para a lógica proposicional, existem tanto sistemas de prova axiomáticos, quanto sistemas de dedução natural que são *corretos e completos*.

1.2.3.1. Exemplo de Sistema de Provas: Sistema Axiomático (*à la Frege-Hilbert*)

Existem diversos sistemas de prova para lógica proposicional e de primeira ordem. Entre as diversas formas de apresentação de sistemas lógicos, sistemas axiomáticos estão entre as mais utilizadas. Esta apresentação também é conhecida como apresentação *à la Frege-Hilbert*, ou, no “estilo de Hilbert” [37]. O sistema axiomático é historicamente muito utilizado por filósofos e matemáticos. Tem como limitação (en. “drawback”) o fato de levar à construção de provas complexas, mesmo para experientes. Um sistema axiomático consiste de fórmulas que são definidas como axiomas e pela especificação de suas regras de inferência. Para exemplificarmos a sua utilização, apresentamos a seguinte axiomatização da lógica proposicional. Se α , β e γ são quaisquer fórmulas bem-formadas de L , então os seguintes são axiomas de L [37]; esta axiomatização utiliza apenas os conectivos \rightarrow , \neg .

- (A1). $(\alpha \rightarrow (\beta \rightarrow \alpha))$
- (A2). $((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)))$
- (A3). $((\neg\beta \rightarrow \neg\alpha) \rightarrow ((\neg\beta \rightarrow \alpha) \rightarrow \beta))$.

Uma regra de inferência de L : *modus ponens*: $\alpha, \alpha \rightarrow \beta \vdash \beta$

Exemplo simples: prova em L de $A \rightarrow A$.

1. $(A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow ((A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A))$ (Instância Ax. A2)
2. $A \rightarrow ((A \rightarrow A) \rightarrow A)$ Esquema Ax. A1
3. $(A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A)$ 1, 2, MP
4. $A \rightarrow (A \rightarrow A)$ Esquema Ax. A1
5. $A \rightarrow A$ A partir de 3, 4 por MP

Há mais de uma forma de axiomatizar a lógica proposicional. O sistema axiomático proposto por Hilbert para a lógica proposicional inclui a regra de inferência modus ponens e os seguintes axiomas:

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$
4. $\neg\neg A \rightarrow A$
5. $A \rightarrow \neg\neg A$

Embora a apresentação axiomática seja sintética, a construção de provas neste sistema é notadamente mais difícil do que em outros sistemas de prova.

1.2.3.2. Exemplo de Sistema de Provas: Sistema de Dedução Natural

O método de dedução natural, foi proposto por Gentzen [23], e refinado por Fitch [16] em um formato onde provas e subprovas são representadas hierarquicamente. O método consiste em uma representação formal da argumentação direta, em que utilizamos regras do sistema para manipular premissas e obter conclusões. A formalização do raciocínio é feita através de regras ditas de *introdução* e de *eliminação*. Regras de *introdução* permitem introduzir em uma linha de prova uma fórmula que utilizam um determinado conetivo que está sendo “introduzido”. Regras de *eliminação* permitem a inferência de uma nova fórmula em que um conetivo foi “eliminado” (estas noções são detalhadas a seguir). Nossa representação utilizará o formato de “boxes” (caixas) que lembram a hierarquia da estrutura original no formato de árvores como Gentzen utilizava. Os livros [3, 4, 19, 29] utilizam os formatos de caixas, que - de certa forma, ressalte-se - lembram também a estrutura de um programa de computador. O método de dedução natural é muito utilizado como formalização do raciocínio utilizado em provas por especialistas; i.e. existe uma correspondência próxima com a forma de como teoremas são provados (na “prática” lógico-matemática) e como são construídas as relações entre premissas e conclusões em dedução natural. Em vez de axiomas, há regras de inferência, que descrevem o que pode ser derivado a partir de premissas e hipóteses.

Em síntese, o sistema de dedução natural tem uma série de vantagens em relação a outros sistemas, como em relação ao sistema axiomático, pois: (i) concentra-se na noção de dedução, que é a noção fundamental em lógica; (ii) evita que tenhamos de definir um conjunto de fórmulas como axiomas; (iii) freqüentemente as regras de dedução natural são mais fáceis de serem manipuladas. (iv) de certa forma, ele espelha a forma como os seres humanos realizam inferências.

As Regras de Dedução do Sistema de Dedução Natural

A implicação material e uso do raciocínio hipotético: Nas ciências exatas o raciocínio hipotético é fundamental. Existem inúmeras publicações em filosofia da ciência sobre o tema, e neste curso apresentamos algumas formalizações elementares sobre o mesmo. Lembre-se que lógica é fundamentalmente baseada sobre o raciocínio condicional (“se... então...”) e este padrão de raciocínio constitui a espinha dorsal das derivações (algumas não triviais) obtidas através de raciocínios hipotéticos. O método de dedução natural ilustra muito bem a formalização metódica deste tipo de raciocínio, fundamental na ciência. Nas regras que manipulam a implicação material “ \rightarrow ”, a negação “ \neg ”, a disjunção “ \vee ” e nas provas por contradição, a utilização de hipóteses é essencial. Hipóteses são fórmulas utilizadas de uma forma distinta das demais fórmulas. Usa-se suposições (através de raciocínios em situações *hipotéticas*) para construir uma situação

na qual esta fórmula é válida. Ou seja, assumimos que em uma determinada situação uma hipótese (fórmula) é válida a fim de obter alguma conclusão a partir desta hipótese (fórmula). Por que usamos hipóteses? De forma simplificada, para derivar informações adicionais (em subprovas) sobre a prova maior que estamos tentando construir.

Em dedução natural, uma das **regras para o condicional** \rightarrow faz uso de hipóteses. Assim, a regra de introdução da implicação material ($\rightarrow I$) é explicada a seguir: Para introduzirmos (i.e. mostrarmos) uma fórmula $\alpha \rightarrow \beta$ em uma linha de prova, assume-se α e então constrói-se a prova de β . Pode-se utilizar α e as demais fórmulas anteriormente derivadas nesta subprova. No entanto, ao obter β , a hipótese α **não** pode ser utilizada novamente, pois esta era uma hipótese adicional para provar β . Esta prova é isolada em uma caixa para indicar que a hipótese é válida apenas “localmente”. Note que hipóteses são anotadas entre colchetes “[α]”. *Eliminação da implicação material*: Para “eliminar” a implicação material em $\alpha \rightarrow \beta$ é necessário também provar α , e assim podemos derivar β . Dizemos “eliminação” por estarmos deduzindo uma fórmula β (que é o consequente de uma implicação $\alpha \rightarrow \beta$). Esta regra também é conhecida como *modus ponens* e é usualmente abreviada em livros de lógica como MP. A seguir, apresentamos as regras de introdução e eliminação da implicação material (representadas por $\rightarrow I$ e $\rightarrow E$, respectivamente).

1. $[\alpha]$ hipótese 2. : 3. β 4. $\alpha \rightarrow \beta \rightarrow I(1,3)$	1. $\alpha \rightarrow \beta$ 2. α 3. $\beta \rightarrow E(1,2)$
--	---

Regras para a conjunção \wedge : *Regra de Introdução*: Para introduzir (ou para escrever) uma fórmula do tipo $\alpha \wedge \beta$ em uma prova, devemos provar α e provar β . *Regra de Eliminação*: Se provamos $\alpha \wedge \beta$ então podemos escrever α e/ou podemos também escrever β . As regras podem ser representadas como a seguir.

1. α 2. : 3. β 4. $\alpha \wedge \beta \wedge I(1,3)$	1. $\alpha \wedge \beta$ 2. $\alpha \wedge E(1)$ 3. $\beta \wedge E(1)$
---	---

Regras para a disjunção \vee : *Regra de Introdução*: Para provar $\alpha \vee \beta$, basta provar α ou basta provar β . (A rigor, há duas regras de introdução do \vee : uma a partir de α e outra para eliminação de β). Note que α, β são fórmulas quaisquer da linguagem proposicional. *Regra de Eliminação*: Para provar algo a partir de $\alpha \vee \beta$, deve-se prová-lo a partir da hipótese que α é válida e a partir da hipótese que β é válida. Esta construção é uma argumentação por casos.

1. α 2. $\alpha \vee \beta \quad \vee I(1)$	1. $\alpha \vee \beta$ 2. $[\alpha]$ hipótese 3. \vdots ⟨prova⟩ 4. γ 8. $\gamma \quad \vee E(1,2-4,5-7)$	5. $[\beta]$ hipótese 6. \vdots ⟨prova⟩ 7. γ
---	---	---

Regras para (\neg): *Introdução:* Para mostrar $\neg\alpha$, assumimos α e provamos \perp . A hipótese não pode ser utilizada fora da subprova. Note que nesta regra, usa-se também o raciocínio hipotético, e interpreta-se $\neg\alpha$ como $\alpha \rightarrow \perp$. *Eliminação:* A partir de α e de $\neg\alpha$, provamos \perp .

1. $[\alpha]$ hipótese 2. \vdots 3. \perp 4. $\neg\alpha \quad \neg I(1,3)$	1. α 2. \vdots 3. $\neg\alpha$ 4. $\perp \quad \neg E(1,3)$
--	---

Entre as regras derivadas do sistema de dedução natural, podemos citar a regra de eliminação da dupla negação $\neg\neg E$: A partir de $\neg\neg\alpha$ provamos α . Esta regra é utilizada para simplificar passos de prova, mas pode ser derivada de outras regras do sistema de dedução natural. A seguir mostramos que, em lógica clássica, *qualquer fórmula pode ser obtida a partir de uma prova de \perp (também referido como falsum)*. Isto é, mostramos que $\perp \vdash A$ para qualquer fórmula A .

1. \perp Premissa 2. $[\neg A]$ hipót. 3. $\perp \checkmark (1)$ 4. $\neg\neg A \quad \neg I(2,3)$ 5. $A \quad \neg\neg E(4)$

Ou seja, ao assumirmos uma situação na qual \perp é verdadeiro (linha 1), se $\neg A$ é verdadeiro (linha 2), então \perp é verdadeiro, o que não é possível. Portanto, temos $\neg\neg A$ (linha 4) e consequentemente, A é verdadeira nesta situação. Isto é: “*qualquer coisa*” (fórmula) segue de uma contradição. Na prova acima, anotamos uma linha com o símbolo “ \checkmark ”. Esta anotação simplesmente justifica a utilização de uma fórmula já provada anteriormente (esta notação é utilizada por alguns autores, incluindo [3]).

Note que é impossível que \perp seja verdadeira. \perp só é provado através de hipóteses contraditórias, e isto acontece no meio de uma prova maior. Em uma subprova, podemos então deduzir aquilo que necessitamos/desejamos. Claramente, mostrar contradições, i.e. mostrar \perp em uma sub-prova, pode ser útil em construção de provas.

Prova por Contradição⁸ Para provar α , assume-se $\neg\alpha$ e prova-se \perp . Novamente, esta é uma regra que utiliza raciocínio condicional (hipotético).

1. $[\neg\alpha]$ hipótese
2. :
3. \perp
4. $\alpha \quad \text{PC}(1,3)$

Regras para \perp : *Introdução:* Para provar \perp temos de provar α e provar $\neg\alpha$, para fórmulas quaisquer. Esta regra é análoga à regra $\neg E$. *Eliminação:* A partir de uma prova de \perp podemos derivar qualquer fórmula.

1. $\neg\alpha$		
2. :	1. \perp	
3. α	2. $\alpha \quad \perp E(1)$	
4. $\perp \quad \perp I(1,3)$		

Existem também regras para \top , de introdução e eliminação. \top pode ser introduzido em qualquer ponto de uma prova, pois é uma fórmula sempre válida, enquanto que se temos \top nada podemos afirmar a partir desta fórmula.

Utilização de Lemas: Lemas são fórmulas provadas (válidas), e assim denominadas quando utilizadas na construção de uma prova maior. Por exemplo, um lema muito utilizado em dedução natural para lógica clássica é a lei do meio excluído, i.e. $\alpha \vee \neg\alpha$. Este lema - em geral - deve ser utilizado em conjunto com uma eliminação de \vee , onde $\neg\alpha$ e α serão hipóteses para um raciocínio por casos.

Regras para \leftrightarrow : Lembre-se que $\alpha \leftrightarrow \beta$ é equivalente a $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$, e também é equivalente a $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$. *Introdução:* Para provar $\alpha \leftrightarrow \beta$, devemos provar α e β ou devemos provar $\neg\alpha$ e $\neg\beta$. Devido à equivalência acima, também podemos derivar $\alpha \leftrightarrow \beta$ ao provarmos $(\alpha \rightarrow \beta)$ e $(\beta \rightarrow \alpha)$. *Eliminação:* A partir de $\alpha \leftrightarrow \beta$, podemos provar $(\alpha \rightarrow \beta)$ e $(\beta \rightarrow \alpha)$ e $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$. Assim como as demais regras para os conetivos, as regras para \leftrightarrow também estão representadas na Figura 1.6.

1. α	
2. :	
3. β	
4. $\alpha \leftrightarrow \beta \leftrightarrow I(1,3)$	
	1. $\alpha \leftrightarrow \beta$ 2. $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta) \leftrightarrow E(1)$ 3. $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \leftrightarrow E(1)$

A seguir, apresentamos a prova de um sequente simples, representada na Figura 1.5. Na prova do sequente $p \rightarrow (q \rightarrow r) \vdash p \wedge q \rightarrow r$, escrevemos a premissa $p \rightarrow (q \rightarrow r)$

⁸Também conhecida como redução ao absurdo (Reductio ad absurdum - RAA).

Figura 1.5. Exemplo: Prova do seqüente $p \rightarrow (q \rightarrow r) \vdash p \wedge q \rightarrow r$ através do sistema de dedução natural, ilustrada nesta figura.

1.	$p \rightarrow (q \rightarrow r)$ premissa
2.	$[p \wedge q]$ hip. $p \quad \wedge E(2)$ $q \quad \wedge E(2)$ $q \rightarrow r \rightarrow E(1,3)$ $r \quad \rightarrow E(4,5)$
3.	
4.	
5.	
6.	
7.	$p \wedge q \rightarrow r \rightarrow I(1,6)$

na linha 1 da Figura 1.5. O nosso objetivo é provar $p \wedge q \rightarrow r$ (que está na linha 7). A conclusão a ser provada é uma implicação material, $\alpha \rightarrow \beta$. Para provar uma implicação material, assumimos como hipótese o antecedente (α) da implicação: portanto, na linha 2 assumimos $[p \wedge q]$ com o objetivo de mostrarmos r . Note que r ocorre como consequente de uma implicação, na linha 1. Assim, será necessário usar a regra de eliminação do condicional $\rightarrow E$ (modus ponens) para mostrarmos r . Isto é feito sucessivamente: primeiramente mostramos p , na linha 3 e posteriormente mostramos q na linha 4. Ao mostrarmos estes antecedentes das implicações da linha 1, finalmente podemos mostrar q , a partir de duas aplicações de modus ponens (linhas 5 e 6). Note que a prova da Figura 1.5 não é desenvolvida apenas de “cima para baixo”. Uma analogia muitíssimo simples sobre como construir uma prova nos remete à construção de caminhos entre dois pontos em um mapa. Ou seja, a construção de uma prova muitas vezes é análoga à construção de um caminho: temos de saber o ponto de saída (premissas, hipóteses) e o ponto de chegada (a conclusão, i.e. a fórmula que temos de provar). A partir destes pontos construímos o “caminho” (a prova) entre estes “pontos”, usando este raciocínio, recursivamente, até que o caminho (prova) esteja concluído.

As regras aqui apresentadas permitem que todas as fórmulas válidas da lógica clássica proposicional sejam provadas usando o sistema de dedução natural. Esta propriedade é conhecida como *completude* de um sistema de provas. Um sistema lógico é completo se, e somente se, todas as fórmulas bem formadas válidas são teoremas do sistema. Idealmente, um sistema de provas teria que permitir que todas as fórmulas válidas

sejam deriváveis através das regras do sistema. Por outro lado, é desejável também que as fórmulas provadas através do sistema de prova sejam efetivamente fórmulas válidas do sistema lógico em questão. Esta propriedade é conhecida como correção de um sistema de provas. O sistema de dedução natural apresentado aqui é correto e completo em relação à semântica apresentada. Isto garante que o sistema de dedução natural pode ser utilizado para verificar a validade de fórmulas proposicionais clássicas. A seção acima resume regras básicas do sistema de dedução natural para a lógica clássica proposicional. Estas regras seriam diferentes para outros sistemas lógicos. Na lógica intuicionista, por exemplo, não se admitem provas por contradição, nem eliminação da dupla negação $\neg\neg E$ e não é aceita a lei (ou o princípio) do meio excluído $\alpha \vee \neg\alpha$, considerado válido na lógica clássica.

Figura 1.6. Resumo das Regras para os Conjetivos Clássicos. As linhas da tabela servem como referência. Obviamente poderíamos adotar uma numeração genérica para as linhas de prova; mas como exemplo didático, a numeração a seguir é clara.

1. α . 2. : 3. β 4. $\alpha \wedge \beta$ $\wedge I(1,3)$	1. $\alpha \wedge \beta$ 2. α $\wedge E(1)$ 3. β $\wedge E(1)$								
1. $[\alpha]$ hipótese . 2. : 3. β 4. $\alpha \rightarrow \beta$ $\rightarrow I(1,3)$	1. $\alpha \rightarrow \beta$ 2. α 3. β $\rightarrow E(1,2)$ 4. $\alpha \rightarrow \beta$ $\rightarrow I(1,3)$								
1. α 2. $\alpha \vee \beta$ $\vee I(1)$	1. $\alpha \vee \beta$ <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">2. $[\alpha]$ hipótese</td> <td style="width: 50%;">5. $[\beta]$ hipótese</td> </tr> <tr> <td>3. :</td> <td>6. :</td> </tr> <tr> <td>\langle prova \rangle</td> <td>\langle prova \rangle</td> </tr> <tr> <td>4. γ</td> <td>7. γ</td> </tr> </table> 8. γ $\vee E(1,2-4,5-7)$	2. $[\alpha]$ hipótese	5. $[\beta]$ hipótese	3. :	6. :	\langle prova \rangle	\langle prova \rangle	4. γ	7. γ
2. $[\alpha]$ hipótese	5. $[\beta]$ hipótese								
3. :	6. :								
\langle prova \rangle	\langle prova \rangle								
4. γ	7. γ								
1. $[\alpha]$ hipótese . 3. \perp 4. $\neg\alpha$ $\neg I(1,3)$	1. α . 3. $\neg\alpha$ 4. \perp $\neg E(1,3)$								
1. $\neg\alpha$. 3. α 4. \perp $\perp I(1,3)$	1. \perp 2. α $\perp E(1)$								
1. $\neg\neg\alpha$ 2. α $\neg\neg E(1)$	1. $[\neg\alpha]$ hipótese . 3. \perp 4. α $PC(1,3)$								
1. α . 3. β 4. $\alpha \leftrightarrow \beta$ $\leftrightarrow I(1,3)$	1. $\alpha \leftrightarrow \beta$ 2. $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$ $\leftrightarrow E(1)$ 3. $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ $\leftrightarrow E(1)$								

1.2.4. O Problema da Satisfatibilidade (SAT)

O problema da satisfatibilidade - SAT - consiste em determinar se existe uma interpretação que satisfaça uma fórmula da lógica proposicional. Dada uma fórmula α existe uma atribuição v tal que $v(\alpha) = 1$ (ou *true*)? Embora o enunciado seja “simples” e extensivamente estudado, apresenta um dos grandes desafios da Ciência da Computação. Até hoje, não há algoritmo em tempo polinomial para o problema. Steven A. Cook⁹ identificou a relevância de SAT em um clássico artigo, publicado no início da década de 1970 [6]. O artigo é intitulado *The Complexity of Theory Proving Procedures*. O próprio título já indica a forte relevância que as questões de lógica tem e sua relação com a área de complexidade computacional. Neste artigo, Cook formalizou as noções de redução em tempo polinomial e mostrou que SAT é um problema NP-completo. Neste artigo, também foi formulada famosa questão *P* versus *NP*, problema que ainda está em aberto.



Figura 1.7. Edgar (Ted) F. Codd, Stephen A. Cook

1.2.5. Lógica de Primeira Ordem

A lógica proposicional, em termos de poder de expressão, é mais “fraca” que a lógica de predicados de primeira ordem. Por exemplo, a afirmação (proposição) “um herói é alguém admirado por todos” é representada simplesmente como uma proposição em lógica proposicional, digamos p ; note que, por exemplo, o predicado “admirado” e o quantificador “todos” são abstraídos na representação. A lógica de predicados aumenta o poder de expressão da lógica proposicional ao adicionar a possibilidade de expressarmos relações entre objetos, inclusive fazendo uso de quantificadores: \forall - que representa a quantificação universal e \exists - que representa a quantificação existencial, além de outras extensões à linguagem. Símbolos de relação n-ários, símbolos de função, constantes e quantificadores sobre variáveis são introduzidos. Na sintaxe, define-se a noção de *assínatura*, que consiste que um conjunto de constantes, símbolos de função e símbolos de relação. A noção de *termo* é fundamental para nomear objetos (utilizamos constantes,

⁹Cook recebeu o ACM A.M. Turing Award em 1982 pelo seu trabalho pioneiro em complexidade computacional. Este prêmio, o mais importante da Ciência da Computação, tem esta denominação em reconhecimento às contribuições de Alan M. Turing.

variáveis, símbolos de função n-ários como termos). Exemplos de fórmulas atômicas na lógica de primeira ordem, construídas com símbolos de relação unária *homem* e de relação binária *amigo*, constantes (*Pedro*, *Celina*, *Luis*), variáveis (x, y) e símbolo de função unária f : $homem(Pedro)$, $amigo(Celina, Luis)$, $\forall x \exists y (y = f(x))$). Note que constantes e símbolos de função se referem a indivíduos, a objetos, enquanto que os símbolos de relação especificam, representam relações entre estes objetos. Esta especificação de relações entre objetos não é representada de forma precisa na lógica proposicional. Ademais, a semântica da lógica de predicados é mais complexa. Neste curso, apresentamos apenas uma noção muito simplificada desta semântica, por questões de espaço.

Para que possamos interpretar esta linguagem mais expressiva, a semântica deve considerar quantificadores, variáveis e novas fórmulas atômicas, que envolvem relações. A noção de *estrutura* ou *modelo* torna-se fundamental. Uma estrutura identifica uma coleção de objetos em seu domínio, bem como o significado dos símbolos da linguagem. A interpretação de uma constante corresponde a uma constante no domínio da estrutura (ou do modelo); a interpretação de um símbolo de relação em uma estrutura corresponde a uma relação sobre os objetos do domínio desta estrutura. Uma *sentença* é uma fórmula sem ocorrência de variáveis livres, e.g. $\forall x \exists y (y = f(x))$; as variáveis deste exemplo estão sob o escopo dos quantificadores \forall e \exists , são ditas amarradas (ou limitadas, ou ligadas) ou ainda *bound* - em inglês. Por sua vez a fórmula $\forall x (P(x, y) \rightarrow R(x, y))$ não é uma sentença, pois a variável y é livre: y não ocorre no escopo de um quantificador \forall ou \exists . Variáveis livres são interpretadas através de uma atribuição de valor para um objeto do domínio da estrutura. De forma muito sucinta, fórmulas quantificadas existencialmente são interpretadas em uma estrutura de forma que alguma atribuição torne a fórmula verdadeira, enquanto que para fórmulas quantificadas universalmente todas as atribuições tornam a fórmula verdadeira.¹⁰

Portanto, a lógica de primeira ordem permite especificar propriedades de estruturas como (grafos, ordens parciais, grupos, bancos de dados) usando predicados (relações). As sentenças atômicas da lógica de predicados têm argumentos (*termos*) que denotam objetos e símbolos de predicados de n argumentos. A utilização de variáveis quantificadas pelo quantificador existencial ($\exists x$) e universal ($\forall x$) permite grande expressividade sobre conjuntos de objetos. Por exemplo, podemos expressar propriedades sobre grados, $G = (V, E)$, $E \subseteq V^2$, como *Cada nodo do grafo tem pelo menos dois vizinhos distintos*: $\forall x \exists y \exists z (\neg(y = z) \wedge E(x, y) \wedge E(x, z))$. Outra observação simples, mas poderosa é que uma *estrutura relacional* é essencialmente um banco de dados relacional. Este “insight” de Ted Codd (no final da década de 1960) posteriormente levou ao desenvolvimento de uma indústria bilionária em bancos de dados relacionais. Codd posteriormente foi agraciado com o ACM Turing Award.

¹⁰Por questões de espaço a apresentação formal da semântica da lógica de primeira ordem não é apresentada neste texto. No entanto, pode ser compreendida através da leitura dos excelentes livros [7, 29].

1.2.5.1. Dedução Natural para Lógica de Predicados

As provas no cálculo de dedução natural para lógica de predicados são similares às provas para lógica proposicional. Introduz-se regras para os quantificadores e para o símbolo de igualdade. As regras para os conetivos são preservadas: as regras para lógica proposicional continuam válidas para a lógica de predicados. Novamente, nossa apresentação sumariza os sistemas apresentados em [4, 19, 29] que apresentam um sistema de dedução natural para lógica de predicados com estrutura hierárquica no formato de caixas. Primeiramente, temos as regras para igualdade ($=$). A reflexividade da igualdade permite escrever em qualquer linha de prova que $t = t$. Isto é, qualquer *termo* t é igual a si mesmo, o que na verdade é um axioma. Dada uma fórmula arbitrária α , se provamos $\alpha(t)$ e provamos que $t = u$, sendo t, u termos, então pela substituição de termos iguais = *sub* podemos mostrar $\alpha(u)$. As regras estão representadas na Figura 1.8. Exemplo: Podemos mostrar, usando as regras acima que (exemplo de [29]):

1. \vdots 2. $t = t$ =refl	1. $\alpha(t)$ provado 2. \vdots 3. $t = u$ provado 4. $\alpha(u)$ =sub (1,3)
---------------------------------	--

Figura 1.8. Regras de Dedução Natural para Igualdade (=)

$$x+1 = 1+x, (x+1 > 1) \rightarrow (x+1 > 0) \vdash (1+x) > 1 \rightarrow (1+x) > 0$$

- $(x + 1) = (1 + x)$ premissa
 - $(x + 1 > 1) \rightarrow (x + 1 > 0)$ premissa
 - $(1 + x > 1) \rightarrow (1 + x > 0)$ =sub(1,2)

Regras para o Quantificador Universal (\forall)

Para qualquer fórmula $\forall x\alpha(x)$, podemos derivar $\alpha(t)$ para qualquer termo fechado t . Isto se deve ao fato de afirmações universais serem verdade para quaisquer objetos de um domínio, isto é para quaisquer termos fechados. Assim, a regra de eliminação do quantificador universal denotada por $\forall E$ pode ser representada como na Figura 1.9. A regra de introdução do quantificador universal $\forall I$ requer a utilização de raciocínio hipotético. É necessário provar, para um termo arbitrário t , ainda não utilizado em nenhuma outra linha de prova, que $\alpha(t)$ é válido. Note que nesta regra escrevemos um termo em uma linha de prova, o que não ocorre nas demais regras do sistema de dedução natural. Note que t é arbitrário, não sendo utilizado na prova posteriormente. A anotação feita à direita $t - \forall I$, ao lado do termo t na linha 1, denota este termo arbitrário t , para o qual temos de provar a fórmula $\alpha(t)$, conforme a Figura 1.9.

Regras para o Quantificador Existencial \exists

Para mostrarmos uma sentença $\exists x\alpha$ através do sistema de dedução natural, temos que mostrar que $\alpha(t)$ para um termo fechado t (que não inclui variáveis). Assim, a regra de introdução do quantificador existencial $\exists I$ é representada na Figura 1.9. A eliminação do quantificador existencial exige a utilização de uma análise de casos, através de raciocínio hipotético. Lembre-se que a quantificação existencial $\exists x\alpha(x)$ significa que α é verdadeira para pelo menos uma atribuição de x ; isto é $\exists x\alpha(x)$ pode ser interpretado como uma disjunção. Se provamos a sentença $\exists x\alpha(x)$, a partir da hipótese de que $\alpha(t)$ é verdadeiro construímos uma prova de β , então podemos afirmar que β foi derivado a partir de $\exists x\alpha(x)$. No formato hierárquico que estamos utilizando, a regra é representada como a seguir. Como exemplo, provamos o seguinte sequente: $\exists x\neg\alpha(x) \vdash \neg\forall x\alpha(x)$.

1. t $t - \forall I$ 2. \vdots 3. $\alpha(t)$ 4. $\forall x\alpha(x) \quad \forall I(1,3)$	1. $\forall x\alpha(x)$ 2. $\alpha(t) \quad \forall E(1)$	1. $\alpha(t)$ 2. $\exists x\alpha(x) \quad \exists I(1)$	1. $\exists x\alpha(x)$ 2. $[\alpha(t)]$ hip. 3. \vdots 4. β 5. $\beta \quad \exists E(1,2-4)$
---	--	--	--

Figura 1.9. Regras de Dedução Natural para os Quantificadores \forall, \exists .

1. $\exists x\neg\alpha(x) \quad \text{premissa}$ 2. $[\forall x\alpha(x)] \quad \text{hip.}$ 3. $[\neg\alpha(t)] \quad \text{hip.}$ 4. $\alpha(t) \quad \forall E(2)$ 5. $\perp \quad \neg E(3,4)$	6. $\perp \quad \exists E(1,3-5)$
	7. $\neg\forall x\alpha(x) \quad \neg I(2,6)$

Explicação: Para provarmos $\neg\forall x\alpha(x)$, temos $\exists x\neg\alpha(x)$ como premissa. Note que a conclusão é uma fórmula negada. Portanto, podemos usar a regra de $\neg I$ para prová-la; fazemos isto assumindo $\forall x\alpha(x)$ como hipótese para derivar uma contradição \perp . Observe que, ao assumirmos $\forall x\alpha(x)$ e tendo a premissa da linha 1 (i.e. $\exists x\neg\alpha(x)$), a hipótese

$\neg\alpha(t)$ da linha 3 pode levar a uma contradição com a fórmula $\alpha(t)$ da linha 4. Assim, obtemos a conclusão.

1.2.5.2. Outros Métodos de Prova

Infelizmente, devido a restrições de espaço, a apresentação de outros métodos avançados, incluindo métodos de prova automática de teoremas é mais adequada para um curso de automação do raciocínio. Por exemplo, o princípio (ou método) da *Resolução* é uma regra de inferência do cálculo de predicados e utilizado extensivamente em inteligência artificial, devido à sua eficiência [31, 32]. O princípio da resolução foi estudado inicialmente por Davis e Putnam na década de 1960. Robinson estendeu o princípio, tornando-o mais eficiente, notadamente por fazer uso de um algoritmo de unificação (detalhes em [32]). A resolução é o mecanismo de execução/automação da linguagem Prolog. Neste método, todas as fórmulas envolvidas devem estar na forma clausal. Para quaisquer duas cláusulas C_1 e C_2 , se existe um literal L_1 em C_1 que é complementar a um literal L_2 em C_2 , então remova L_1 e L_2 de C_1 e C_2 , respectivamente, e construa a disjunção das cláusulas restantes. Ou, se α, β são cláusulas e p_i, q_j são fórmulas atômicas, então a regra é:

$$\frac{\alpha \vee p_1 \vee \dots \vee p_m \quad \beta \vee \neg q_1 \vee \dots \vee \neg q_m}{(\alpha \vee \beta)\theta}$$

onde θ é o *Unificador Mais Geral (UMG)* de todas as fórmulas p_i e q_j . O UMG é obtido por *unificação*; a unificação busca determinar se dois termos coincidem. No caso da lógica proposicional, é possível construir um provador de teoremas utilizando o princípio da resolução que é correto e completo. Uma característica essencial da resolução é o fato deste método funcionar exclusivamente sobre sentenças em *forma clausal* - disjunções de literais (átomos ou negações de átomos). A forma clausal é utilizada por métodos automáticos de prova desde os primórdios desta área. A transformação de uma sentença em forma clausal preserva a consistência (ou inconsistência) da sentença original, e desta forma o uso de cláusulas facilita a busca por refutações. Os métodos não-clausais também fazem uso das idéias sugeridas para melhorar o desempenho de provadores de teoremas clausais.

Outro método de inferência relevante é o conhecido como *tableaux analítico*, desenvolvido inicialmente por Beth e aprimorado por Smullyan [7, 45]. É um método refutacional. Ou seja, para provarmos o sequente $\beta_1, \dots, \beta_n \vdash \alpha_1, \dots, \alpha_m$, afirmamos os antecedentes β_1, \dots, β_n em busca de uma contradição; se a contradição é mostrada pelo tableau, o sequente está provado.

Os assuntos relacionados à prova automática de teoremas são de alta complexidade, demandando conhecimento de diversas lógicas, algoritmos e complexidade computacional. A automação do raciocínio tem uma longa história. Ainda na década de 1950, Martin Davis construiu um programa para automatizar a aritmética de Presburger (a teoria de primeira ordem dos naturais com adição). Allan Newell, Herbert Simon and Cliff Shaw desenvolveram o “Logic Theorist”: um programa que imitava o raciocínio humano, que também foi fundamental nos primórdios da inteligência artificial. Este sistema foi capaz de provar 38 dos 52 primeiros teoremas do *Principia Mathematica* [52]. Na década de 1960, os algoritmos de Davis-Putnam (1960) e o algoritmo DPLL (Davis-Putnam-Logemann-Loveland, 1962) foram desenvolvidos. DPLL é aplicado na resolução de problemas de satisfatibilidade (SAT). Até hoje variações e extensões do algoritmo DPLL são utilizadas pelos modernos “SAT solvers”. Para problemas de SAT em lógica de primeira ordem podemos utilizar SMT (Satisfiability Modulo Theories), que têm obtido sucesso notável na área de verificação, particularmente em aplicações em engenharia de software [13]. A área de projeto de circuitos integrados e aplicações críticas também fazem uso intensivo de métodos automáticos de prova.

1.2.6. Lógicas-Não-Clássicas

Em lógica clássica assumimos uma única situação, um único mundo possível em que fórmulas são avaliadas, isto é, em que assumem valores-verdade. Em diversos domínios do conhecimento, este modelo não é satisfatório. Em Ciência da Computação, nosso conhecimento do mundo, pode se referir a outros “mundos/estados possíveis” [25]. Raciocinar/refletir sobre conhecimento, computações e ações é fundamental em computação: como processos se coordenam para executar uma ação? O que robôs precisam saber para executar suas tarefas? O que programas (“agentes”) precisam saber sobre os outros programas para se comunicarem? As lógicas que investigam estas questões são lógicas não-clássicas: modais, temporais, epistêmicas, entre outras.

Além disso, sistemas computacionais são dinâmicos e interativos. O tempo, por exemplo, é um aspecto fundamental, assim como estados, transições, trocas de mensagens. Para aplicações em Ciência da Computação, Inteligência Artificial entre outras, estas lógicas requerem uma formalização/axiomatização distinta da realizada através de lógica clássica. A lógica clássica é monotônica, i.e. um resultado não é refutado sob novas informações. Em inteligência artificial, no entanto, muitas vezes é necessário utilizar raciocínios não-monotônicos. Tendo em vista as características e a complexidade das aplicações em IA e computação, surgem então diversas propostas de lógicas “não-clássicas”.¹¹

¹¹Além disso, as lógicas não-clássicas encontram um grande respaldo da comunidade de Ciência da Computação, inclusive através de um grande número de Periódicos (Journals), Handbooks, Livros e Conferências. É também notável que um percentual significativo de pesquisadores que atuam em áreas relacionadas à lógica computacional tenham sido vencedores do ACM Turing Award. A lista de premia-

Entre as lógicas originárias da Ciência da Computação podemos citar as lógicas desenvolvidas para raciocinar sobre programas, a partir dos trabalhos de Hoare, Dijkstra e Floyd [29]. Hoare e Milner também desenvolveram cálculos computacionais para expressar propriedades de concorrência e interação, que demandam novas lógicas computacionais para expressar propriedades de sistemas concorrentes [27, 38]. Dijkstra foi um pioneiro da área de verificação formal e da *derivação de programas*, método no qual um programa e sua prova são desenvolvidos concomitantemente. Uma lógica modal que



Figura 1.10. E.W. Dijkstra, C.A.R. Hoare, Robin Milner: Turing Awards (1972, 1980, 1991)

se tornou extremamente útil em ciência da computação é a lógica temporal. O raciocínio sobre tempo é fundamental em computação. Este requisito essencial do ponto de vista computacional, tem conduzido ao desenvolvimento de um considerável número de sistemas lógicos temporais. Estes desenvolvimentos iniciaram-se com o estudo de lógicas modais temporais, com forte impacto desde o trabalho de Pnueli *On the temporal logic of programs*, a partir de 1977 [40]. As lógicas temporais tem mostrado efetiva aplicabilidade na verificação de programas, verificação de hardware, teoria de bancos de dados, computação distribuída. É relevante observar que a lógica LTL (Linear Temporal Logic) proposicional tem o poder expressivo da lógica de primeira ordem sobre os naturais [21].

1.2.6.1. Lógicas Não-Clássicas: Lógicas Modais

Not ignorance, but ignorance of ignorance, is the death of knowledge.
Alfred North Whitehead

As lógicas modais formalizam modos de verdade. Em filosofia, são relacionadas à epistemologia, sendo relevantes no estudo do conhecimento. São as lógicas das verdades contingênciais, do “necessário” e “possível”. Sua origem moderna remonta a origem a Lewis [35]. Há, também, extensões para lógica modal de primeira ordem. Na sintaxe das lógicas modais, adiciona-se inicialmente operadores unários à lógica clássica proposicional. Os operadores básicos \Box , \Diamond adicionados à lógica proposicional/predicados são

dos inclui M. Rabin & D. Scott, C.A.R. Hoare, R. Milner, J. McCarthy, E. Codd, S. Cook, A. Pnueli, E. Clarke, A. Emerson e J. Sifakis, que realizaram trabalhos em áreas diretamente vinculadas à lógica.

$\Box\phi$: que lemos ϕ “é necessariamente verdadeiro”, e $\Diamond\phi$ que é lido “ ϕ é possivelmente verdadeiro” (e equivalente a $\neg\Box\neg\phi$). Em lógica modal uma proposição é necessária em um mundo ω se ela for verdadeira em todos os mundos que são possíveis em relação a este mundo ω , enquanto que uma proposição é possível em um mundo se ela for verdadeira em pelo menos um mundo possível alternativo, em relação a ω . Esta relação é formalizada através de uma relação binária de acessibilidade entre mundos possíveis. As lógicas modais foram consideradas apropriadas para estudar provas (lógica de provas), tempo, conhecimento, crenças, obrigações e outras modalidades [4, 15]. Em inteligência artificial e computação, lógicas modais estão entre os formalismos mais adequados para análise e representação de conhecimento e raciocínio sobre sistemas distribuídos e concorrentes [15, 18].

Em aplicações em epistemologia, e em Ciência da Computação, na modelagem de conhecimento em sistemas distribuídos e em inteligência artificial, $\Box\phi$ é lido como “de acordo com o meu conhecimento”, “de acordo com as leis da física”, “após o término do programa”. Notadamente a partir da década de 1950, os filósofos da lógica (e epistemologistas) Hintikka e Kripke propuseram a *semântica de mundos possíveis* [26, 34]. A partir da década de 1970, as lógicas modais passaram a ser muito utilizadas em Ciência da Computação na modelagem e o raciocínio sobre tempo, espaço, crenças, ações e conhecimento, individualmente ou através de combinações destas. Uma obra de referência que tornou as lógicas modais relativamente populares em computação (dizemos relativamente, pois o estudo de lógica computacional reúne uma comunidade científica pequena) foi o agora clássico “Reasoning about Knowledge” de Fagin, Halpern, Moses, Vardi [15]. Ademais, as propriedades de decidibilidade da lógica modal motivam seu uso em computação. Vardi mostra que elas são robustamente decidíveis, sendo adequadas, portanto, para aplicações em computação. [51].



Figura 1.11. Saul Kripke, Amir Pnueli, Moshe Vardi

A lógica modal encontra diversas aplicações. Diversas leituras (ou interpretações) das modalidades são possíveis, por exemplo:

- Modalidades Aléticas:
 - $\Box\phi$: ϕ é necessariamente verdadeiro.
 - $\Diamond\phi$: ϕ é possivelmente verdadeiro (equivalente a $\neg\Box\neg\phi$).

- Modalidades epistêmicas

$K_a\phi$: Agente a sabe ϕ .

$B_a\phi$: Agente a acredita que ϕ é verdade.

- Modalidades temporais lineares:

$\circ\phi$ (ou $X\phi$): ϕ será verdade no próximo ponto de tempo (estado do mundo).

$\square\phi$ (ou $G\phi$): ϕ será verdadeiro sempre.

$\diamond\phi$ (ou $F\phi$): ϕ será possivelmente (contingencialmente) verdadeiro no futuro (equivalente a $\neg\Box\neg\phi$: ϕ não será sempre falso).

$\phi\Box\psi$: ϕ é verdadeiro até que ψ seja verdadeiro.

A axiomatização básica da lógica modal acrescenta novos axiomas à lógica proposicional. O Sistema Axiomático da lógica modal conhecido como sistema K , consiste dos axiomas abaixo e duas regras de inferência:

A1 Todas as tautologias proposicionais.

A2 (**K**) $\Box(\alpha \rightarrow \beta) \rightarrow (\Box\alpha \rightarrow \Box\beta)$

R1 De α , $\alpha \rightarrow \beta$ infira β (MP)

R2 De α infira $\Box\alpha$ (Generalização)

Kripke demonstrou, na década de 1960, que o sistema K é um sistema axiomático correto e completo em relação ao modelo semântico. O axioma A2 é hoje chamado de **K** em homenagem a Kripke. Os sistemas lógicos modais ditos normais contém o axioma **K** e a regra de generalização acima. Quanto a exemplos de fórmulas, podemos citar: $\Box\Box A$: é necessário que A seja necessário (ou “é conhecido que A é conhecido”); $\Box\Diamond A$: é necessário que A é possível; $\Box A \rightarrow A$: se A é conhecido, então A é verdadeiro (ou se A é necessariamente verdadeiro, então A é verdadeiro). É importante observar que a cada propriedade da relação de acessibilidade corresponde um axioma da lógica modal. Por exemplo, a seguinte lista apresenta alguns axiomas da lógica modal e a correspondência com as correspondentes propriedades da relação binária R .

T: $\Box\alpha \rightarrow \alpha$ $\forall xR(x,x)$ (Reflexiva)

4: $\Box\alpha \rightarrow \Box\Box\alpha$ $\forall x,y,z(R(x,y) \wedge R(y,z)) \rightarrow R(x,z)$ (Transitiva)

B: $\alpha \rightarrow \Box\Diamond\alpha$ $\forall x,y(R(x,y) \rightarrow R(y,x))$ (Simétrica)

D: $\Box\alpha \rightarrow \Diamond\alpha$ $\forall x\exists yR(x,y)$ (Serial)

5: $\Diamond\alpha \rightarrow \Box\Diamond\alpha$ $\forall x,y,z(R(x,y) \wedge R(x,z)) \rightarrow R(y,z)$ (Euclidiana)

Assim, obtemos diferentes sistemas de lógicas modais considerando subconjuntos destes axiomas. Por exemplo:

K: A1, A2, R1, R2.

KD45: K, D, 4, 5 + A1, R1, R2.

KT4: tipicamente chamado de **S4**

KT45: tipicamente chamado de **S5**.

O axioma T corresponde à propriedade em R da reflexividade; 4 corresponde à propriedade da transitividade; B corresponde à propriedade da simetria; D corresponde à propriedade serial e 5 corresponde à propriedade euclidiana.

1.2.6.2. Semântica de Mundos Possíveis

Embora existam diversos modelos semânticos para as lógicas modais, esta síntese se concentra na semântica relacional de Kripke/Hintikka. O modelo dos mundos possíveis de Kripke é utilizado de forma ampla, inclusive como modelo de conhecimento, para lógicas modais epistêmicas. Kripke [33, 34] definiu uma teoria de modelos baseada em “mundos possíveis” e relações de acessibilidade explícita entre mundos. Estes modelos são aplicáveis em uma classe ampla de lógicas. A noção de que mundos são “possíveis” depende do mundo atual, ou de referência. Por exemplo, em lógica modal temporal as noções de passado e futuro dependem do mundo atual. Se considerarmos um conjunto W de mundos possíveis e uma relação binária R em W , podemos ler $R(\omega_1, \omega_2)$ como ω_2 é um mundo possível tendo ω_1 como referência, ou a partir de ω_1 . Podemos considerar diversas propriedades das relações de acessibilidade. Um *frame* de Kripke é definido por uma estrutura (W, R) onde W é um conjunto de não vazio e R é uma relação em W .

A noção de relação de acessibilidade também é relevante pois sistemas diferentes de lógicas modais se caracterizam pelas propriedades das relações de acessibilidade. Por exemplo, se definirmos que a relação de acessibilidade é reflexiva, simétrica e transitiva definimos um sistema lógico conhecido como lógica modal $S5$ (ou $KT45$) [15]. De forma simplificada, podemos definir então uma Estrutura de Kripke: $M = (W, R, v)$, onde: W é um conjunto de mundos possíveis; $R \subseteq W^2$ é a relação de acessibilidade entre mundos; e $v(\omega)$ associa a cada mundo $\omega \in W$ um conjunto de variáveis proposicionais. A satisfação de fórmulas $(M, \omega) \models \alpha$ (que pode ser lida α é verdadeira em (M, ω) ou α é satisfeita em (M, ω)), pode então ser definida:

- $(M, \omega) \models p$ se, e somente se, $p \in v(\omega)$.
- $(M, \omega) \models \neg\alpha \iff (M, \omega) \not\models \alpha$
- $(M, \omega) \models \alpha \wedge \beta \iff (M, \omega) \models \alpha$ and $(M, \omega) \models \beta$
- $(M, \omega) \models \alpha \vee \beta \iff (M, \omega) \models \alpha$ or $(M, \omega) \models \beta$
- $(M, \omega) \models \alpha \rightarrow \beta \iff (M, \omega) \models \beta$ or $(M, \omega) \not\models \alpha$

- $(M, \omega) \models \Box\varphi \iff$ para todo $\omega_i \in W$, se $R(\omega, \omega_i)$ então $(M, \omega_i) \models \varphi$
- $(M, \omega) \models \Diamond\varphi \iff$ existe ω_i tal que $R(\omega, \omega_i)$ e $(M, \omega_i) \models \varphi$

Observe que a definição de satisfação para os conetivos clássicos reflete a definição da lógica proposicional clássica. Em cada mundo possível, a lógica proposicional clássica segue a sua interpretação usual. A lógica modal é uma extensão da lógica clássica pois um modelo da lógica modal é uma estrutura de modelos da lógica clássica interdependentes de acordo com a satisfatibilidade das fórmulas modais. A noção de *validade* também é mais geral. A validade pode ser definida em relação a um dado modelo ou em relação a uma dada classe de modelos. Diferentes funções de avaliação para um mesmo frame (W, R) podem gerar um classe C de modelos associados a (W, R) . Dizemos então que, para um modelo $M = (W, R, v)$ e α uma fórmula bem-formada, α é *válida* no modelo M , denotado por $M \models \alpha$, se para todo ω em W , então $M, \omega \models \alpha$ (i.e. α é verdadeiro em todo mundo possível de W). A fórmula α será válida em uma classe C de modelos se para todo modelo M' de C , temos $M' \models \alpha$.

Também é possível considerar múltiplos agentes em uma lógica modal. Neste caso, cada agente tem a sua própria visão de mundo, expresso por uma relação de acessibilidade individual. Cada agente teria uma modalidade \Box_i , indexada pelo agente, além de uma relação R_i . Em uma lógica epistêmica, podemos utilizar uma modalidade K para representar conhecimento. Assim $K_i\alpha$ significa que o agente i sabe α ; $K_2K_1\neg\alpha$ pode ser lido: o agente 2 sabe que o agente 1 sabe $\neg\alpha$. Por exemplo, o modelo a seguir representando três relações, expressaria um modelo de Kripke para a lógica $KT45^n$, com três agentes (exemplo de [29]). A Figura 1.12 representa o Modelo M a seguir (não mostramos reflexividade, transitividade na figura). $M = \langle W = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}; R_1 = \{(\omega_1, \omega_2), (\omega_1, \omega_3), (\omega_3, \omega_2), (\omega_4, \omega_5)\}; R_2 = \{(\omega_1, \omega_3), (\omega_4, \omega_5)\}, R_3 = \{(\omega_3, \omega_2), (\omega_5, \omega_6)\}; v = \{(\omega_1, q), (\omega_2, p), (\omega_2, q), (\omega_3, p), (\omega_4, q), (\omega_6, p)\} \rangle$

1.2.6.3. Sistemas de Provas Para Lógicas Modais

Vários tipos de sistemas de prova para lógicas modais foram propostos, e.g., [4, 17]. Em alguns destes sistemas, as fórmulas são rotuladas pelos mundos (estados) em que elas são verdadeiras, o que pode facilitar no entendimento e desenvolvimento das provas. Nas regras apresentadas a seguir, no estilo de dedução natural, a notação $\varphi : \omega$ significa que φ é verdadeira no mundo possível ω . A relação de acessibilidade R também é explicitada neste sistema de dedução rotulado, o que também auxilia na derivação de em que mundos (estados) as fórmulas podem ser inferidas a partir da informação sobre as relações R entre mundos. As diferenças entre as diversas lógicas, por exemplo lógicas modais, são ditadas pelas propriedades diferentes das relações de acessibilidade. Isso é possível pela união, nas *unidades declarativas* que são constituídas de fórmulas com

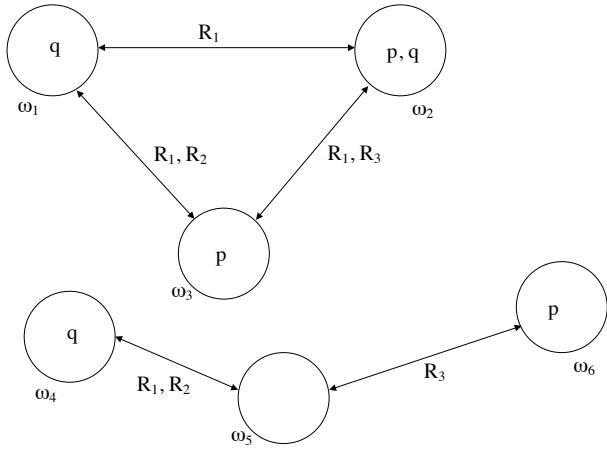


Figura 1.12. Exemplo de Modelo para $KT45^n$ (de [29])

rótulos. Também definimos uma álgebra de rotulação que representa as propriedades específicas de cada sistema lógico. Há também regras de inferência estruturais (não formalizadas aqui por simplicidade) que permitem raciocínio sobre diagramas (relações) em configurações.

Na regra $\Box I$ a hipótese $[\mathcal{R}(\omega, g_\varphi(\omega))]$ é interpretada como: dado um mundo possível arbitrário $g_\varphi(\omega)$, se podemos inferir $\varphi : g_\varphi(\omega)$ então é possível provar que $\Box\varphi : \omega$. A regra $\Diamond E$ pode ser vista, informalmente, como uma *skolemização* do quantificador existencial sobre mundos possíveis, o que é semanticamente implicado pela fórmula $\Diamond\varphi$ na premissa. Assim, o termo $f_\varphi(\omega)$ define um mundo possível particular unicamente associado à fórmula φ inferido e acessível a partir do mundo possível ω (i.e. $\mathcal{R}(\omega, f_\varphi(\omega))$). A regra de $\Diamond I$ representa que se tivermos a relação $\mathcal{R}(\omega_1, \omega_2)$ e se φ é verdade em ω_2 então derivamos $\Diamond\varphi$ no mundo ω_1 . Finalmente, a regra $\Box E$ significa que, se $\Box\varphi$ vale no mundo ω_1 e ω_1 está relacionado com (ω_2 é acessível de) ω_2 então podemos inferir que φ é verdadeiro em ω_2 . Assim, as fórmulas são rotuladas pelos mundos em que elas são verdadeiras para facilitar o processo de raciocínio e inferência. As referências explícitas às relações de acessibilidade também auxiliam na derivação de fórmulas nos mundos relacionados por R [4].

Tabela 1.4. Regras para operadores modais

$\frac{[R(\omega, g_\varphi(\omega))]}{\varphi : \omega}$ \vdots $\frac{\varphi : g_\varphi(\omega)}{\square\varphi : \omega} \square I$	$\frac{\square\varphi : \omega_1, R(\omega_1, \omega_2)}{\varphi : \omega_2} \square E$ $\frac{\varphi : \omega_2, R(\omega_1, \omega_2)}{\diamond\varphi : \omega_1} \diamond I$
$\frac{\diamond\varphi : \omega}{\varphi : f_\varphi(\omega), R(\omega, f_\varphi(\omega))} \diamond E$	

Exemplo: Vamos provar $\square p : w_0 \vdash \neg\diamond\neg p : w_0$ usando regras de dedução rotuladas.

1. $C_1 \langle \square p : w_0 \vdash \neg\diamond\neg p : w_0 \rangle$ premissa

2. $C_2 \langle \square p : w_0, [\diamond\neg p : w_0] \rangle$ hip.

3. $C_3 \langle \square p : w_0, R(w_0, f_{\neg p}(w_0)), \neg p : f_{\neg p}(w_0) \rangle$ $\diamond E(C_2)$

4. $C_4 \langle p : f_{\neg p}(w_0), \neg p : f_{\neg p}(w_0) \rangle$ $\square E(C_2)$

5. $C_5 \langle \perp : f_{\neg p}(w_0) \rangle$ $\wedge I(C_3)$

6. $C_6 \langle \neg\diamond\neg p : w_0 \rangle$ $\neg I(C_2, C_5)$

Explicação: Nesta prova, temos como premissa $\square p : w_0$, isto é no mundo w_0 , $\square p$ é verdadeiro. Para provar a conclusão $\neg\diamond\neg p : w_0$, assumimos, como hipótese que $\diamond\neg p : w_0$ para obtermos uma contradição \perp usando a regra de $\neg I$ em w_0 . Note que, em w_0 , temos agora $\square p : w_0$, $\neg\diamond\neg p : w_0$ e $\diamond\neg p : w_0$ (hipótese). Para obtermos uma contradição, temos de usar as regras de eliminação $\square E$, quanto $\diamond E$. Para usar $\diamond E$, temos mostrar que p é verdade em um novo mundo possível unicamente associado com p , isto é em $f_{\neg p}(w_0)$. Com a contradição entre $p : f_{\neg p}(w_0)$ e $\neg p : f_{\neg p}(w_0)$, concluímos a prova. Note que neste sistema rotulado de provas, fazemos uso das regras para os conetivos clássicos, conforme a Tabela 1.5.

1.2.6.4. Lógica em Inteligência Artificial: Integrando Raciocínio e Aprendizado

Em 2003, Valiant propôs um grande desafio para a Ciência da Computação [48]. O desafio consiste no desenvolvimento de modelos efetivos de computação cognitiva inte-

Tabela 1.5. Regras para conetivos clássicos

$\frac{\begin{array}{c} [\alpha : \omega] & [\beta : \omega] \\ \vdots & \vdots \\ \alpha \vee \beta : \omega & \gamma : \omega & \gamma : \omega \end{array}}{\gamma : \omega} \vee E \quad \frac{\alpha : \omega}{\alpha \vee \beta : \omega} \vee I$	$\frac{\alpha \wedge \beta : \omega}{\alpha : \omega, \beta : \omega} \wedge E \quad \frac{\alpha : \omega, \beta : \omega}{\alpha \wedge \beta : \omega} \wedge I$
$\frac{\alpha \supset \beta : \omega, \alpha : \omega}{\beta : \omega} \supset E \quad \frac{\begin{array}{c} [\alpha : \omega] \\ \vdots \\ \beta : \omega \end{array}}{\alpha \supset \beta : \omega} \supset I$	$\frac{\neg \neg \alpha : \omega}{\alpha : \omega} \neg E \quad \frac{\begin{array}{c} [\alpha : \omega] \\ \vdots \\ \perp : \omega' \end{array}}{\neg \alpha : \omega} \neg I$

grando raciocínio e aprendizado.

“The aim here is to identify a way of looking at and manipulating common-sense knowledge that is consistent with and can support what we consider to be the two most fundamental aspects of intelligent cognitive behaviour: the ability to learn from experience, and the ability to reason from what has been learned. We are therefore seeking a semantics of knowledge that can computationally support the basic phenomena of intelligent behaviour.”[48]

Recentemente, esta área tem sido objeto de pesquisas de um crescente número de pesquisadores, tendo em vista resultados promissores em inteligência artificial [8]. Para responder efetivamente a este desafio, além de conhecimentos sólidos de ciência da computação, é necessário entendimento da ciência cognitiva, área de estudo interdisciplinar da mente e cérebro. Esta área envolve psicologia, filosofia, IA, neurociência, linguística, antropologia. Em cognição, estudamos os processos do raciocínio humano, enquanto que em computação estes processos podem ou não ter inspiração na natureza. Várias áreas nos auxiliam a construir artefatos/algoritmos/sistemas computacio-

nais mais expressivos, que podem contribuir neste desafio. Outro desafio, relacionado ao proposto por Valiant é o clássico questionamento levantado por Turing:

I propose to consider the question, “Can machines think?” This should begin with definitions of the meaning of the terms “machine” and “think”. The definitions might be framed so as to reflect so far as possible the normal use of the words,... but this attitude is dangerous, If the meaning of the words “machine” and “think” are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, “Can machines think?” is to be sought in a statistical survey such as a Gallup poll. But this is absurd.... [47].

Para realizar computação cognitiva, diversas habilidades devem ser consideradas. Uma alternativa inicial seria expressar algumas destas habilidades em sistemas artificiais. Estes modelos cognitivos mais completos requerem, portanto, a integração entre aprendizado, formação de regras (lógicas) e raciocínio (computação) [30]. Para construir modelos computacionais cognitivos que exigem a integração de raciocínio expressivo (através de lógicas) e aprendizado robusto é necessário definir uma semântica adequada para computação cognitiva, i.e. que represente habilidades cognitivas.

Como aprendizado de máquina é usualmente estudado através de abordagens estatísticas, experimentais, enquanto que o raciocínio em IA é estudado através de abordagens baseadas em lógicas, uma alternativa seria desenvolver modelos integrados, híbridos, por exemplo, modelos neuro-simbólicos. A computação neuro-simbólica explora os benefícios de cada paradigma: aprendizado (neural) e raciocínio simbólico (lógico) [12]. O exemplo a seguir mostra como modelos neuro-simbólicos podem ser realizados. Nesta abordagem, o conhecimento é representado por uma linguagem lógica simbólica, enquanto que a computação é realizada por algoritmos conexionistas, através de redes neurais. Para ilustrar a abordagem, utilizarmos a Lógica Modal Conexionista (Connectionist Modal Logic - CML) [11]. O *insight* em CML é considerar as redes neurais como redes de mundos possíveis, como na Figura 1.13.

No exemplo, considere que as 3 redes estão relacionadas, podendo se comunicar. Considere as redes neurais N_1, N_2, N_3 como mundos possíveis. Observe que em N_1 é possível, por exemplo, usar o raciocínio sobre a fórmula $\Box q$ para deduzirmos q em N_2 . Por sua vez, em N_2 temos s ; assim, podemos mostrar $\Diamond s$ em N_1 . Outros raciocínios são possíveis. Regras de inferência podem ser aprendidas pela rede neural, integrando, assim, raciocínio lógico e aprendizado de máquina. Observe que estas redes neuro-simbólicas tem propriedades interessantes, inclusive em relação à computação de pontos fixos de programas lógicos. Por exemplo, para qualquer programa P , existe uma rede neural N tal que N computa o operador de ponto fixo de programas modais, de

programas lógicos que representam raciocínio intuicionista e de linguagens restritas de programas temporais [9, 10, 11].

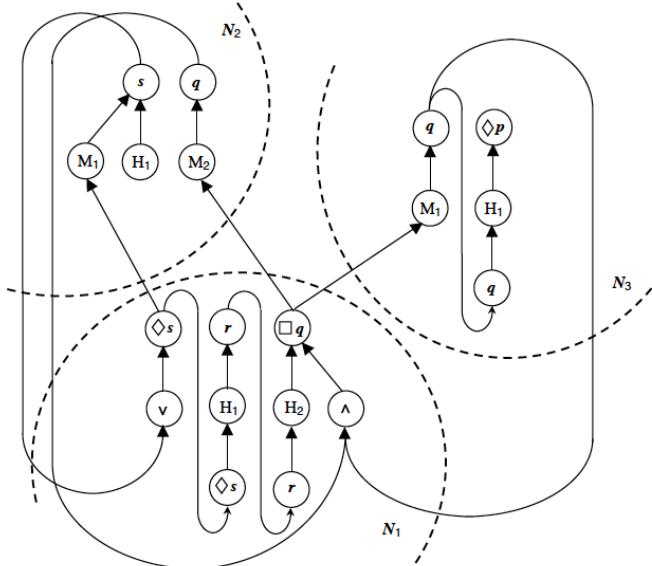


Figura 1.13. Conjunto de redes que representa o programa
 $P = \{r \rightarrow \square q : \omega_1; \diamond s \rightarrow r : \omega_1; s : \omega_2; q \rightarrow \diamond p : \omega_3, R(\omega_1, \omega_2), R(\omega_2, \omega_3)\}$

Outras áreas em que a integração de lógica e aprendizado teve aplicações relevantes incluem raciocínio e aprendizado temporal, inclusive no desenvolvimento de simuladores de treinamento [14], devido à capacidade de aprender novas regras observando-se experts e novos usuários em treinamento. Nos problemas tipicamente utilizados como casos de estudo em sistemas distribuídos (*muddy children puzzle*, *wise men puzzle*, *dining philosophers* [15]), os resultados de aplicação desta metodologia também foram promissores, pois o sistema é capaz de aprender a resolver estes estudos de caso, encontrando a solução dos mesmos (veja detalhes em [12]). Finalmente, em engenharia de software, a abordagem permite adaptação e evolução de modelos e especificações temporais, servindo para verificar sistemas quando descrições de modelos são incompletas [2].

Também é relevante mencionar que a integração entre **lógica e probabilidade** tem sido tema de pesquisas por um longo período. Historicamente, De Morgan e Boole já haviam realizado estudos em que associavam raciocínio probabilístico e lógica clássica. Esta área de pesquisa tem como um de seus objetivos integrar a representação simbólica da lógica - que representa verdades absolutas, com a representação de incertezas - através de probabilidade [24, 28]. Embora este pareça ser um objetivo

contraditório, a combinação de raciocínio qualitativo (simbólico) e quantitativo (probabilístico) pode levar à construção de sistemas, particularmente em inteligência artificial, mais expressivos [41].



Figura 1.14. Michael O. Rabin, Leslie G. Valiant

1.2.7. Conclusões e Perspectivas

O estudo de sistemas lógicos têm sido fundamental desde as origens da Ciência da Computação, a ponto da lógica ser citada como o cálculo da computação [25]. Recentemente, temos presenciado resultados muito promissores de pesquisas em Ciência da Computação e Inteligência Artificial [43, 50], que demonstram que os sistemas computacionais atuais já são capazes de integrar habilidades cognitivas complexas, como o aprendizado a partir de experiências e o raciocínio sobre o conhecimento adquirido do ambiente. A construção de sistemas computacionais com complexidade crescente, exige, portanto, conhecimento sofisticado de técnicas, modelos e fundamentos da Ciência da Computação.

Os grandes desafios científicos sempre exigiram e, cada vez mais, irão demandar habilidade e capacidade de formalizar e construir raciocínios não triviais em sistemas de computação. Este capítulo ofereceu uma introdução mínima e incompleta à lógica aplicada. Esperamos que o leitor, a partir da bibliografia, possa explorar novos caminhos de forma independente. Para concluir de forma promissora, citamos Michael Rabin, vencedor do ACM Turing Award em 1976:

Our field is still in its embryonic stage. It's great that we haven't been around for 2000 years. We are at a stage where very, very important results occur in front of our eyes. M.O. Rabin em [44]

Referências

- [1] Dirk Baltzly. Stoicism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2014.
- [2] Rafael V. Borges, Artur d'Avila Garcez, and Luís C. Lamb. Learning and representing temporal knowledge in recurrent networks. *IEEE Transactions on Neural Networks*, 22(12):2409–2421, 2011.

- [3] K. Broda, S. Eisenbach, H. Khoshnevisan, and S. Vickers. *Reasoned Programming*. Prentice Hall, 1994.
- [4] K. Broda, D.M. Gabbay, L.C. Lamb, and A. Russo. *Compiled Labelled Deductive Systems: A Uniform Presentation of Non-classical Logics*. Studies in Logic and Computation. Research Studies Press/Institute of Physics Publishing, Baldock, UK, Philadelphia, PA, 2004.
- [5] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, April 1936.
- [6] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.
- [7] Flávio Corrêa da Silva, Marcelo Finger, and Ana C. V. de Melo. *Lógica para Computação*. Thomson, São Paulo, 2006.
- [8] Artur d’Avila Garcez, Marco Gori, Pascal Hitzler, and Luís C. Lamb. Neural-Symbolic Learning and Reasoning (Dagstuhl Seminar 14381). *Dagstuhl Reports*, 4(9):50–84, 2015.
- [9] A.S. d’Avila Garcez and L.C. Lamb. A connectionist computational model for epistemic and temporal reasoning. *Neural Computation*, 18(7):1711–1738, 2006.
- [10] A.S. d’Avila Garcez, L.C. Lamb, and D.M. Gabbay. Connectionist computations of intuitionistic reasoning. *Theoretical Computer Science*, 358(1):34–55, 2006.
- [11] A.S. d’Avila Garcez, L.C. Lamb, and D.M. Gabbay. Connectionist modal logic: Representing modalities in neural networks. *Theoretical Computer Science*, 371(1-2):34–53, 2007.
- [12] A.S. d’Avila Garcez, L.C. Lamb, and D.M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Cognitive Technologies. Springer, 2009.
- [13] Leonardo M. de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, 2011.
- [14] Leo de Penning, Artur S. d’Avila Garcez, Luís C. Lamb, and John-Jules Ch. Meyer. A neural-symbolic cognitive agent for online learning and reasoning. In Toby Walsh, editor, *IJCAI-11*, pages 1653–1658. IJCAI/AAAI, 2011.
- [15] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

- [16] Frederic B. Fitch. *Symbolic Logic*. The Ronald Press Company, New York, 1952.
- [17] M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel Publishing Company, Dordrecht, 1983.
- [18] D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*, volume 148 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, 2003.
- [19] D. M. Gabbay. *Elementary Logics: a Procedural Perspective*. Prentice Hall, London, 1998.
- [20] Dov M. Gabbay and F. Guenther, editors. *Handbook of Philosophical Logic*, volume I-XVIII. Springer, 2008-2015.
- [21] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal basis of fairness. In *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*, pages 163–173, 1980.
- [22] Dov M. Gabbay and John Woods, editors. *Handbook of The History of Logic*, volume I-XI. Elsevier, 2008-2015.
- [23] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 1934.
- [24] J.Y. Halpern. *Reasoning about Uncertainty*, MIT Press, 2003.
- [25] J.Y. Halpern, R. Harper, N. Immerman, P.G. Kolaitis, M.Y. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, 7(2):213–236, 2001.
- [26] J. Hintikka. *Knowledge and Belief*. Cornell University Press, 1962.
- [27] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [28] C. Howson. Probability and logic. *J. Applied Logic* 1(3-4), 151-165, 2003.
- [29] M. Huth and M. Ryan. *Logic in Computer Science: modelling and reasoning about systems*. Cambridge University Press, 2000.
- [30] R. Kharden and D. Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, 1997.
- [31] Robert A. Kowalski. *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge University Press, 2011.

- [32] Robert A. Kowalski. *Logic for Problem Solving, Revisited*. Herstellung und Verlag: Books on Demand, 2014.
- [33] S. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–4, 1959.
- [34] S. Kripke. Semantic analysis of modal logics I, normal propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.
- [35] C. Lewis. *A Survey of Symbolic Logic*. University of California Press, Berkeley, 1918.
- [36] Z. Manna and R. Waldinger. *The logical basis for computer programming. Volume 1: deductive reasoning*. Addison-Wesley, Boston, 1985.
- [37] E. Mendelson. *Introduction to mathematical logic*. Van Nostrand Reinhold, New York, 1964.
- [38] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
- [39] Joan Moschovakis. Intuitionistic logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2015 edition, 2015.
- [40] A. Pnueli. The temporal logic of programs. In *Proceedings of 18th IEEE Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.
- [41] S.J. Russell. Unifying Logic and Probability: A New Dawn for AI? In *Information Processing and Management of Uncertainty in Knowledge-Based Systems - 15th International Conference, IPMU 2014*, pages 10–14, 2014.
- [42] Ralph Schoenman, editor. *Bertrand Russell: Philosopher of the Century*. Allen and Unwin, London, 1967.
- [43] Bernhard Schölkopf. Artificial intelligence: Learning to see and act. *Nature*, 518:486–487, 2015.
- [44] Denis Shasha and Cathy Lazere. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, 1995.
- [45] R. M. Smullyan. *First-Order Logic*. Dover Publications, New York, revised edition, 1995.
- [46] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2), 1936.

- [47] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.
- [48] L. G. Valiant. Three problems in computer science. *Journal of ACM*, 50(1):96–99, 2003.
- [49] Dirk Van Dalen. Intuitionistic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume 166 of *Synthese Library*, pages 225–339. Springer Netherlands, 1986.
- [50] Moshe Y. Vardi. Is information technology destroying the middle class? *Commun. ACM*, 58(2):5, 2015.
- [51] M.Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *Discrete Mathematics and Theoretical Computer Science*, pages 149–184. DIMACS, 1997.
- [52] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I,II, III. Cambridge University Press, Cambridge, 1910, 1912, 1913.
- [53] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Kegan Paul, Trench, Tubner & Co, London, 1922.

Capítulo

2

Computação Urbana: Técnicas para o Estudo de Sociedades com Redes de Sensoriamento Participativo

Thiago H. Silva e Antonio A. F. Loureiro

Abstract

Urban computing is a recent research topic that aims to obtain and analyze urban data from various sources, such as traditional wireless sensor networks (WSNs) and emerging participatory sensor networks (PSNs) to understand and address issues that cities face. PSNs are particularly interesting because they rely on the participation of users in urban sensing, allowing the observation of large-scale actions of people in (almost) real time over long periods of time. PSN data increase our knowledge about different aspects of our lives in urban scenarios, which can be very useful in developing more sophisticated applications to various sectors, especially those related to the understanding of urban societies. The purpose of this short course is to discuss the concept of urban computing and urban sensing with participatory sensor networks. We aim to show the relevance of urban computing and motivate the construction of new applications that address issues related to the dynamics of cities and urban social behavior. In addition, this short course will discuss how to work with PSNs, by analyzing their properties and their usefulness in the development of new applications in urban computing.

Resumo

A computação urbana (*urban computing*) é um tema recente de pesquisa que visa obter e analisar dados urbanos de diversas fontes, como as tradicionais redes de sensores sem fio (RSSFs) e as emergentes redes de sensoriamento participativo (RSP), com o objetivo de entender e tratar questões enfrentadas pelas cidades. As RSPs são particularmente interessantes nesse caso, pois contam com a colaboração dos usuários no sensoriamento urbano, permitindo a observação das ações das pessoas em larga escala em tempo (quase) real durante longos períodos de tempo. Dados de RSPs aumentam o nosso conhecimento sobre diferentes aspectos de nossas vidas em cenários urbanos o que pode ser bastante útil no desenvolvimento de aplicações mais sofisticadas para diversos segmentos, principalmente os relacionados com o entendimento de sociedades urbanas. O objetivo deste minicurso é discutir o conceito de computação urbana e de sensoriamento urbano com redes de sensoriamento participativo. Visamos mostrar a relevância da computação urbana e motivar a

construção de novas aplicações que sirvam para tratar questões relacionadas com a dinâmica de cidades e do comportamento social urbano. Além disso, este minicurso discutirá como trabalhar com RSPs, ao analisar as suas propriedades e a sua utilidade no desenvolvimento de novas aplicações na área de computação urbana.

2.1. Introdução

A computação urbana (*urban computing*) [Kindberg et al. 2007, Kostakos and O'Neill 2008, Zheng et al. 2014a] é uma área interdisciplinar que diz respeito ao estudo e tratamento de questões enfrentadas pelas cidades utilizando tecnologia de computação. Por essa razão, a computação urbana conta com profissionais e aplicações em campos que incluem: antropologia, planejamento urbano, engenharia civil, ciência da computação, entre outros.

Como mais de 50% da população do mundo hoje vive em cidades [Martine et al. 2007], uma das consequências é uma enorme pressão sobre as suas infraestruturas, como transporte, habitação, água e energia, gerando difíceis desafios. Para entender e tratar essas e outras questões com o intuito de melhorar a qualidade de vida das pessoas que vivem em cidades, na computação urbana são usadas diversas fontes de dados sobre o ambiente urbano, alguns exemplos incluem: as tradicionais redes de sensores sem fio (RSSFs) [Loureiro et al. 2003]; e as emergentes redes de sensoriamento participativo (RSP) [Burke et al. 2006, Silva et al. 2014a].

As RSPs são particularmente interessantes nesse caso, pois contam com a colaboração dos usuários no sensoriamento urbano e permitem a observação em larga escala das ações das pessoas em tempo (quase) real durante longos períodos de tempo, possibilitando o entendimento da dinâmica da cidade e do comportamento social urbano. Com isso, as RSPs têm o potencial de se tornarem ferramentas fundamentais para a computação urbana. Dados de RSPs aumentam o nosso conhecimento sobre diferentes aspectos de nossas vidas em cenários urbanos o que pode ser bastante útil no desenvolvimento de aplicações mais sofisticadas em diversos segmentos, principalmente os relacionados com o entendimento de sociedades urbanas.

O objetivo deste minicurso é apresentar o conceito de computação urbana e de sensoriamento urbano com o auxílio de redes de sensoriamento participativo. Isso inclui uma visão geral de trabalhos específicos que ilustram as tendências de pesquisa e os principais desafios e oportunidades da área.

O restante do capítulo está organizado da seguinte forma. A Seção 2.2 apresenta o conceito de computação urbana, incluindo um arcabouço para o desenvolvimento de aplicações nessa área e algumas das principais fontes de dados. A Seção 2.3 discute em mais detalhes uma das fontes de dados urbanos: as redes de sensoriamento participativo. A Seção 2.4 discute o gerenciamento de dados urbanos, o que inclui a obtenção e tratamento desses dados. A Seção 2.5 analisa dados urbanos de RSPs, apresentando algumas de suas principais propriedades. A Seção 2.6 apresenta as abordagens e modelos utilizados em diversas aplicações e serviços relacionados ao estudo de sociedades urbanas utilizando dados de RSPs. A Seção 2.7 discute as principais técnicas utilizadas nos trabalhos mencionados nas seções anteriores, bem como algumas das tecnologias e ferramentas comumente utilizadas para a análise de dados. A Seção 2.8 apresenta alguns dos principais desafios relacionados

com a utilização de RSPs na computação urbana, já a Seção 2.9 apresenta várias oportunidades nessa mesma direção. Finalmente, a Seção 2.10 apresenta as nossas conclusões.

2.2. Computação Urbana

2.2.1. Definição

O termo “computação urbana” foi introduzido pela primeira vez por Eric Paulos na edição de 2004 da conferência UbiComp [Eric Paulos and Townsend 2004] e em seu artigo *The Familiar Stranger* [Paulos and Goodman 2004], publicado nesse mesmo ano.

Pode-se definir a computação urbana como um processo de aquisição, integração e análise de um grande volume de dados heterogêneos gerados por diversas fontes em espaços urbanos, tais como sensores, veículos e seres humanos, para ajudar na solução de diversos problemas que as cidades enfrentam tais como congestionamento de trânsito, poluição do ar, falta de água e aumento do consumo de energia. Assim um dos principais objetivos dessa área é ajudar a melhorar a qualidade de vida das pessoas que vivem em ambientes urbanos [Zheng et al. 2014a].

A computação urbana também nos auxilia a compreender a natureza dos fenômenos urbanos, bem como prever o futuro das cidades. Essa é uma área bastante interdisciplinar resultante da fusão da área de ciência da computação com áreas tradicionais, como transporte, economia e sociologia no contexto dos espaços urbanos. No domínio da ciência da computação, a computação urbana tem interseção com, por exemplo, sistemas distribuídos, interação humano-computador, redes de computadores, redes de sensores, sistemas cooperativos e inteligência artificial.

2.2.2. Arcabouço da Computação Urbana

Nesta seção apresentamos um arcabouço para a computação urbana. A Figura 2.1 mostra uma visão geral desse arcabouço, destacando os três componentes mais importantes: (i) gerenciamento dos dados urbanos; (ii) análise dos dados urbanos; e (iii) desenvolvimento de serviços e aplicações.



Figura 2.1. Visão geral do arcabouço da computação urbana.

Como ilustrado na figura, o componente gerenciamento de dados urbanos é composto de alguns passos importantes. O primeiro deles é o processo de coleta de dados urbanos, que podem ser obtidos de diversas fontes de dados, como discutido na próxima se-

ção. O segundo passo refere-se ao processamento desses dados. Após esse processamento podemos modelar os dados em diversos formatos, por exemplo, no formato de grafos, como discutido na Seção 2.4.

O componente análise dos dados urbanos é composto pela etapa de edição e execução de códigos, bem como a interpretação de resultados. Essa parte é fundamental, pois para utilizar dados urbanos é necessário conhecer suas propriedades. Mais detalhes sobre esse componente é descrito na Seção 2.5. Após a etapa de análise, o próximo passo é o desenvolvimento de serviços e aplicações com o conhecimento obtido. Essas aplicações podem ser de diversos tipos, como discutido na Seção 2.6.

2.2.3. Fontes de Dados Urbanos

Nesta seção apresentamos algumas das principais fontes de obtenção de dados urbanos. Esse dados oferecem suporte no desenvolvimento de novos serviços e aplicações na área de computação urbana.

- **Dados estatísticos oficiais:** fornecem dados referentes a um estudo estatístico sobre uma população, tais como dados demográficos, econômicos e sociais relativos a um momento determinado ou em certos períodos.

É possível encontrar diversas fontes de dados na Web disponibilizando dados dessa categoria para algumas localidades, como mostrado em [Barbosa et al. 2014]. No entanto, nem sempre esses dados estão disponíveis para a localidade que se deseja estudar. Outra dificuldade é a diversidade dos formatos nos quais os dados estão disponíveis, como em tabelas, mapas, gráficos, calendários, formulários, entre outros [Barbosa et al. 2014].

- **Redes de sensores tradicionais:** fornecem dados que são obtidos através da instalação de sensores específicos para algumas aplicações, por exemplo, sensores de presença em ruas e avenidas para detectar o volume de tráfego nesses locais, sensores para monitoramento da qualidade do ar em diversos pontos da cidade ou sensores para o monitoramento de níveis de ruídos.

Um problema com essa fonte de dados é a dificuldade de acesso aos dados. Além do custo de construção de uma rede de sensoriamento, geralmente, a implantação de sensores na cidade só é permitida pela prefeitura.

- **Infraestrutura das cidades:** fornecem dados que são capturados aproveitando as infraestruturas existentes da cidade, que são criadas para outros propósitos. Por exemplo, as redes de telefonia celular são construídas para comunicação móvel entre os indivíduos. No entanto, os sinais dos telefones celulares de um grande número de pessoas podem ser usados para tentar predizer a mobilidade dos usuários e melhorar o planejamento urbano.

Outros tipos incluem a localização de veículos que possuem GPS. É cada vez mais comum ônibus, táxis, e veículos privados possuírem GPS embutidos. Esse tipo de dado contribui, por exemplo, para o entendimento do tráfego de uma cidade. Além disso, é possível obter dados de utilização do sistema de transporte público, já que é

bem comum esse tipo de sistema utilizar cartões RFID para registrar o uso de ônibus e metrô dos usuários.

A dificuldade de acesso a dados é também um problema dessa fonte, uma vez que somente a prefeitura ou empresas responsáveis, tipicamente, possuem acesso a esse tipo de dado.

- **Redes de sensoriamento participativos:** fornecem dados urbanos, que possuem uma escala bastante abrangente e podem ser mais fáceis de obter do que as outras fontes mencionadas, pois contam com a colaboração dos usuários na coleta de dados. Além disso, as RSPs podem possuir uma rede social online o que permite o estudo da estrutura social dos usuários, como relacionamentos e interações entre os usuários.

2.3. Sensoriamento Urbano com Redes de Sensoriamento Participativo

Como apresentamos na Seção 2.2, existem várias formas de obter dados urbanos, dentre elas podemos citar as emergentes redes de sensoriamento participativo (RSPs) [Silva et al. 2014a, Burke et al. 2006]. O tema será abordado da seguinte maneira: a Seção 2.3.1 apresenta a definição de uma RSP; a Seção 2.3.2 discute o funcionamento de uma RSP, enquanto a Seção 2.3.3 ilustra exemplos de RSPs.

2.3.1. O que é uma rede de sensoriamento participativo?

O sensoriamento participativo pode ser definido como um processo distribuído de coleta de dados pessoais e sobre diversos aspectos da cidade. Tal processo requer a participação ativa das pessoas para compartilhar voluntariamente informação contextual e/ou tornar seus dados sensoriados disponíveis [Burke et al. 2006], ou seja, o usuário determina manualmente como, quando, o quê e onde amostrar. Assim, através das RSPs é possível monitorar diversos aspectos das cidades, bem como o comportamento coletivo de pessoas conectadas à Internet em tempo (quase) real.

As RSPs têm se tornado populares graças ao aumento do uso de dispositivos portáteis, como *smartphones* e *tablets*, assim como a adoção mundial de sites de mídia social. Com isso, um elemento central de uma rede de sensoriamento participativo é a existência de um usuário capaz de realizar um sensoriamento, por exemplo, da cidade, com um dispositivo computacional portátil. Nesse cenário, as pessoas participam como sensores sociais, fornecendo dados voluntariamente sobre um determinado aspecto de um local que implicitamente capturam as suas experiências de vida diária. Esses dados podem ser obtidos com a ajuda de dispositivos de sensoriamento como, por exemplo, sensores incorporados a *smartphones* (e.g., GPS, acelerômetro, microfone, e outros) ou por meio de sensores humanos (e.g., visão). Nesse último caso, os dados são observações subjetivas produzidas pelos usuários [Silva et al. 2014a, Burke et al. 2006].

As RSPs oferecem oportunidades sem precedentes de acesso a dados de sensoriamento em escala planetária. Essa grande quantidade de dados facilita a obtenção de informações que não estão disponíveis prontamente com a mesma abrangência global, podendo ser usadas para melhorar os processos de tomada de decisão de diferentes entidades (e.g., pessoas, grupos, serviços, aplicações).

Vale ressaltar que vários termos definidos recentemente como, por exemplo, *Humans as Data Sources* e *Ubiquitous Crowdsourcing*, refletem basicamente a definição de redes de sensoriamento participativo [Srivastava et al. 2012, Mashhadi and Capra 2011, Ganti et al. 2011]. É importante também mencionar que o termo sensoriamento oportunista [Lane et al. 2010], que denomina uma forma de sensoriamento que também utiliza dispositivos móveis dos usuários no processo de sensoriamento, pode gerar confusão com o termo sensoriamento participativo. O sensoriamento participativo difere de sensoriamento oportunista principalmente pela participação do usuário, onde, neste último tipo, a etapa de coleta de dados é automatizada, sem a participação do usuário [Lane et al. 2008, Lane et al. 2010].

O sensoriamento oportunista apoia o processo de sensoriamento de uma aplicação sem demandar esforços do usuário, determinando automaticamente quando os dispositivos podem ser usados para atender às demandas específicas das aplicações. Desta forma, os aplicativos podem aproveitar as capacidades de sensoriamento de todos os dispositivos dos usuários do sistema sem a necessidade de intervenção humana neste processo [Lane et al. 2008].

2.3.2. O funcionamento de uma RSP

De forma similar às tradicionais redes de sensores sem fio (RSSFs) [Loureiro et al. 2003], o dado sensoriado em uma RSP é enviado para o servidor, ou “nó sorvedouro”, onde os dados podem ser acessados (usando, por exemplo, APIs, como a API do Instagram¹). Mas, diferentemente das RSSFs, as RSPs têm as seguintes características: (a) nós sensores são entidades móveis autônomas, ou seja, uma pessoa com um dispositivo móvel; (b) o custo da rede é distribuído entre os nós sensores, proporcionando uma escalabilidade global; (c) o sensoriamento depende da vontade das pessoas participarem desse processo; e (d) nós sensores não possuem severas limitações de energia.

As RSPs têm o potencial para complementar as RSSFs em diversos aspectos. As tradicionais redes de sensores sem fio foram projetadas para sensoriar áreas de tamanho limitado, como florestas e vulcões. Em contrapartida, as RSPs podem alcançar áreas de tamanhos variados e de larga escala, como grandes metrópoles, países ou até mesmo todo o planeta [Silva et al. 2014a]. Além disso, uma RSSF está sujeita a falhas, uma vez que o seu funcionamento depende da correta coordenação das ações dos seus nós sensores que possuem severas restrições de energia, processamento e memória. Já as RSPs são formadas por entidades autônomas e independentes, os seres humanos, o que torna a tarefa de sensoriamento mais robusto a falhas individuais. Obviamente, RSPs trazem também vários novos desafios como, por exemplo, o seu sucesso está diretamente ligado à popularização dos *smartphones*, *tablets* e serviços de mídia social.

A Figura 2.2 ilustra uma RSP constituída de usuários com seus dispositivos móveis enviando dados sensoriados sobre suas localizações para sistemas na nuvem. A figura mostra as atividades de compartilhamento (representados por pontos na nuvem) de quatro usuários em três instantes diferentes no tempo, rotulados como “Tempo 1”, “Tempo 2” e “Tempo 3”. Note que um usuário não participa, necessariamente, no sistema em todos os instantes. Após um certo tempo, podemos analisar estes dados de diferentes maneiras. Por exemplo, a parte inferior mais à direita da figura mostra, por meio de

¹<http://instagram.com/developer>.

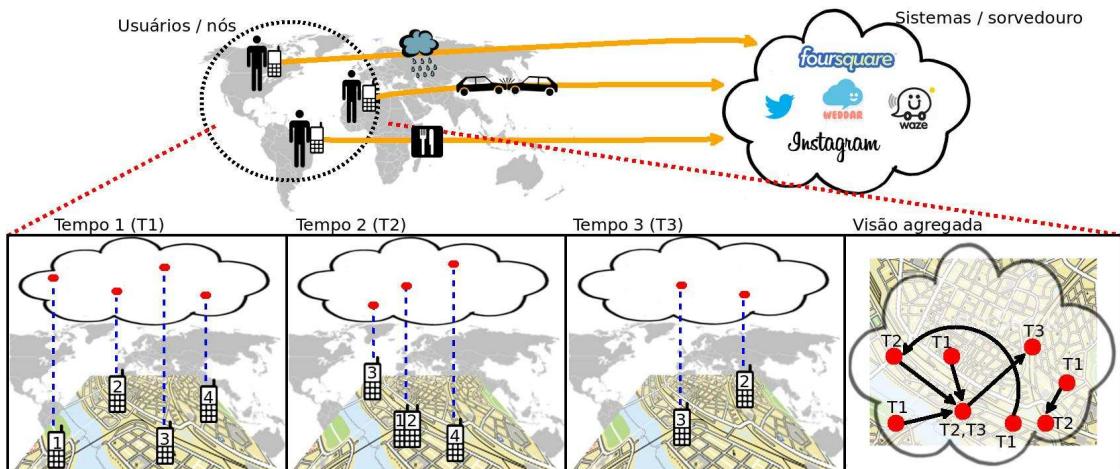


Figura 2.2. Ilustração de uma rede de sensoriamento participativo [Silva et al. 2014a].

uma visão agregada, um grafo dirigido em que os nós/vértices representam os locais onde os dados foram compartilhados e com arestas que conectam localidades que foram compartilhadas pelo mesmo usuário. Usando este grafo podemos extrair, por exemplo, padrões de mobilidade dos usuários que podem ser utilizados para efetuar um gerenciamento de carga de forma mais eficiente na infraestrutura urbana de redes sem fio. Na verdade, a descoberta de conhecimento em RSPs caminha junto com o uso da teoria de grafos/redes [Easley and Kleinberg 2010, Newman 2010, Newman 2003].

2.3.3. Exemplos de RSPs

As redes sociais baseadas em localização, que são um tipo especial de mídia social que combinam características de rede social *online*² e a possibilidade de compartilhar dados com informações espaço-temporais³, podem ser consideradas os exemplos mais populares de RSPs. É possível encontrar vários exemplos de tais sistemas em funcionamento, tais como: Waze, que serve para relatar condições de tráfego em tempo real; Foursquare, para compartilhar o local onde o usuário está visitando; e Instagram, para enviar imagens em tempo real para o sistema. Em particular, o Instagram pode ser visto como uma das mais populares RSPs atualmente, com 200 milhões de usuários [Instagram 2014]. Ao considerarmos essa rede, o dado sensoriado é uma foto de um lugar específico. Podemos extrair informação desse tipo de dado de diversas maneiras. Uma das possibilidades é visualizar em tempo real como está a situação de uma certa área da cidade. Outras possibilidades são discutidas na Seção 2.6.

Note que todos os sistemas descritos anteriormente são compostos de uma rede social *online*. No entanto, existem vários exemplos de RSPs que não contêm redes sociais *online*. Por exemplo, o Weddar⁴, para relatar condições meteorológicas, o NoiseTube⁵, para

²Plataforma virtual que constroe e reflete as relações sociais da vida real entre as pessoas.

³Tipo de dado que permite, por exemplo, a construção de serviços baseados em localização.

⁴<http://www.weddar.com>.

⁵<http://noisetube.net>.

o compartilhamento de nível de ruído em determinada região da cidade ou o Colab⁶, para o compartilhamento de problemas diversos das cidades. Além desses exemplos, podemos citar também o GarbageWatch [CENS/UCLA], para monitorar aspectos do lixo de uma cidade. Repare ainda que a utilização da Web também não é mandatória em uma RSP. Os dados sensoriados podem ser enviados para uma aplicação específica que esteja fora da Web.

2.4. Gerência de Dados Urbanos

2.4.1. Obtenção de Dados

Nesta seção apresentamos três das principais formas de obtenção de dados de RSPs. A primeira forma é através de APIs, como descrita na Seção 2.4.1.1. A segunda forma é utilizando um Web *crawler* (Seção 2.4.1.2). Por fim, apresentamos na Seção 2.4.1.3 a forma de obtenção de dados que utiliza aplicações.

2.4.1.1. Utilizando APIs

A Web está repleta de fontes de informação, o que representa uma grande oportunidade para pesquisadores de diversas áreas coletarem dados em larga escala e a partir deles extrair conhecimento [Benevenuto et al. 2011].

Algumas RSPs disponibilizam APIs que podem ser utilizadas para a extração de dados. Através desse processo, é possível obter dados de RSPs que podem ser utilizados em outras aplicações ou em análises específicas. Várias RSPs populares, como Foursquare, possuem APIs de acesso aos dados compartilhados pelos usuários. Entretanto, é comum existirem regras diferentes para a sua utilização.

Podemos citar duas formas de funcionamento de APIs: (1) baseadas em *streaming*; (2) baseadas em requisições. O método baseado em *streaming* permite coletar em tempo (quase) real os dados que são publicados em uma determinada RSP. A API de *streaming* do Twitter⁷, por exemplo, permite coletar em tempo (quase) real *tweets* públicos à medida que são publicados. Já o método baseado em requisições disponibilizam dados atendendo a uma solicitação específica, por exemplo, todos os seus últimos 10 *tweets*. Tanto métodos baseados em *streaming* quanto métodos baseados em requisições podem sofrer limitações na obtenção do volume de dados. Por exemplo, o Flickr permite 3600 requisições por hora em sua API, já a API de *streaming* do Twitter pode não fornecer todos os dados compartilhados⁸. Isso pode inviabilizar alguns tipos de análises que necessitam de um número maior de amostras no período de uma hora, por exemplo.

De fato, o uso de APIs é uma forma bastante popular para a obtenção de dados. Dados obtidos através de APIs como a do Twitter foram utilizados das mais variadas formas, desde medir a influência de usuários na rede [Cha et al. 2010], até a previsão de terremotos [Sakaki et al. 2010a].

Um exemplo de uso da API de *streaming* do Twitter, escrito na linguagem de progra-

⁶www.colab.re.

⁷<http://www.twitter.com>.

⁸Ao solicitar dados os dados compartilhados no Twitter estima-se que será entregue 1%.

mação Python e utilizando a biblioteca TwitterAPI⁹, é mostrado no algoritmo mostrado na Figura 2.3. Nesse algoritmo fazemos acesso aos *tweets* buscando pela palavra-chave "4sq". Como podemos ver, em poucas linhas de código é possível coletar facilmente dados do Twitter. A Figura 2.4 ilustra esse resultado com dois *tweets* de resposta: tweet1 e tweet2. Esses *tweets* foram retornados no padrão JSON.

```
#Biblioteca que facilita a interação com a API DO Twitter
from TwitterAPI import TwitterAPI

#Um registro no website da API fornece as credenciais indicadas aqui
twitter_api = TwitterAPI(consumer_key='xxxxxx', consumer_secret='xxxxxx',
    access_token_key='xxxxxxxx', access_token_secret='xxxxxxxx')

filters = {"track": ["4sq"]} #palavra que deseja buscar em tweets

stream = twitter_api.request('statuses/filter', filters)

for item in stream.get_iterator():
    print item #exibe todo conteúdo do tweet
```

Figura 2.3. Exemplo de obtenção de dados do Twitter.

Existem RSPs que possuem APIs, mas com acesso bastante restrito aos dados. Esse é o caso do Foursquare, pois poucos dados são possíveis de serem coletados sem a autorização do usuário. A maioria dos dados disponíveis através dessa API são referentes aos locais, como dicas, listas, localização e fotos.

Essas limitações estimulam a obtenção de dados de forma indireta ou alternativa. Por exemplo, em [Silva et al. 2014c] os autores obtiveram dados sobre os *check-ins* do Foursquare através de mensagens públicas compartilhadas no Twitter. Isso é possível, pois o Foursquare possibilita aos usuários anunciar seus *check-ins* nesse sistema. Esse procedimento é mostrado na Figura 2.4. Essa figura ilustra um tweet proveniente do Foursquare que possui uma URL que representa uma página Web com mais informações sobre o *check-in* anunciado. No exemplo da figura, a página representa um *check-in* dado em um restaurante. Para obter mais dados sobre o *check-in* disponíveis nessa página é utilizada a técnica de coleta Web *crawler*, apresentada a seguir.



Figura 2.4. Etapas de coleta de dados do Foursquare através de tweets.

2.4.1.2. Utilizando um Web Crawler

Nem todas as fontes de dados disponíveis na Internet fornecem acesso direto a esses dados através de APIs. Por isso é necessário utilizar outras formas de obtenção de dados. Uma

⁹<https://github.com/geduldig/TwitterAPI>.

dessas alternativas é a chamada Web *crawler*, que são programas que analisam páginas Web em busca de dados relevantes [Benevenuto et al. 2011]. Um Web *crawler* funciona como um robô que acessa páginas Web predeterminadas e recupera dados a partir dessas páginas.

A coleta através de Web *crawlers* depende da estrutura da fonte da qual desejamos obter dados, bem como da abordagem utilizada. A estrutura da fonte é onde os dados que queremos extrair estão disponibilizados; nas páginas Web, por exemplo, são *tags* HTML que apresentam os dados ao usuário. Com isso a construção de um Web *crawler* demanda tipicamente a mineração de texto para a extração dos dados necessários na página Web estudada. No entanto, outras formas não convencionais de extração de dados usando páginas Web são possíveis. Por exemplo, em [Tostes et al. 2014] os autores construíram um Web *crawler* para coletar informações de tráfego tirando fotos (*screenshots*) de mapas com essas informações, como as disponíveis no Bing Maps¹⁰. Mais informações sobre esse procedimento são fornecidas em [Tostes et al. 2014].

```
import urllib
#url obtida através do tweet Saída do código
url = "http://4sq.com/1mhXj6u"
pagina = urllib.urlopen(url).read()
print pagina
```

```
{
  "venue": {
    "name": "Baskent Cafe",
    "stats": {
      "checkinsCount": 7585,
      "usersCount": 14097
    },
    "city": "Pendik",
    "crossStreet": "Ankara",
    "address": "Ankara Cad., Pendik",
    "cc": "TR",
    "id": "59acf230e4bfbbae22beaecc",
    "canonicalUrl": "http://59acf230e4bfbbae22beaecc",
    "name": "Cafe"
  }
}
```

Figura 2.5. Exemplo de coleta de uma página de um check-in do Foursquare usando a biblioteca URLLIB da linguagem Python.

A Figura 2.5 ilustra um código de um simples Web *crawler* em Python. Esse Web *crawler* utiliza a biblioteca URLLIB para realizar a coleta de uma página referente a um check-in do Foursquare (a URL utilizada foi a encontrada no processo ilustrado na Figura 2.4). O resultado parcial da saída do código é também ilustrado nessa figura. Repare que foi assinalada uma informação relevante sobre o check-in: o tipo do local, informação que não é acessível através da API do Foursquare.

2.4.1.3. Utilizando Aplicações

Uma outra alternativa para a coleta de dados é a criação de aplicações em plataformas já existentes. Alguns sistemas populares, como Facebook, Instagram e Runkeeper, permitem a criação de aplicativos dentro de suas plataformas. Com isso, desenvolvedores podem oferecer serviços utilizando dados que são compartilhados nesses aplicativos.

O Facebook, por exemplo, não permite a coleta de informações direta de seus usuários por APIs ou Web *crawlers*. No entanto, como permitem a criação de aplicações é possível obter dados compartilhados por seus usuários. Quando o usuário do Facebook instala um aplicativo e autoriza a leitura de seus dados, o desenvolvedor da aplicação pode ler e armazenar diversos dados, como os disponibilizados pelos usuários, por exemplo, o conteúdo compartilhado com seus amigos.

Em [Nazir et al. 2008] os autores utilizaram essa abordagem de coleta de dados. Eles criaram aplicações do Facebook especificamente para coletar dados que possibilitassem o

¹⁰<http://www.bing.com/maps>.

estudo do comportamento das pessoas que fazem uso desse tipo de aplicação. Outro exemplo foi o aplicativo utilizado em [Youyou et al. 2015]. Os autores criaram uma aplicação no Facebook que captura os últimos *likes*¹¹ do usuário para traçar um perfil de personalidade.

É possível ainda a criação de aplicações que não dependem de plataformas de sistemas existentes. Esse foi o caso da RSP NoiseTube [Maisonneuve et al. 2009]. Os autores criaram uma aplicação que permite aos usuários reportarem níveis de ruído na cidade. Esses dados permitem identificar, por exemplo, quais áreas da cidade o nível de ruído está acima dos limites estipulados por lei. Outro exemplo é o Colab, citado anteriormente. Recentemente foi proposta uma plataforma chamada *ohmage*¹² para facilitar a construção de aplicações que desejam utilizar dados de sensoriamento participativo.

Dessa forma, de posse de dados de RSPs, que podem ser obtidos por alguma dessas maneiras citadas, podemos extrair conhecimento de diversas formas, como é melhor discutido nas próximas seções

2.4.2. Reformatação e Limpeza dos Dados

Os dados brutos (sem tratamento) de RSPs podem não estar em um formato conveniente para executar uma análise particular. Dependendo do tipo do dado é possível encontrar erros semânticos, entradas ausentes ou formatação inconsistente. Nestes casos, eles precisam ser “limpos” antes da análise.

Programadores reformatam e limpam dados escrevendo *scripts* ou editando manualmente dados, por exemplo, em uma planilha. Estas tarefas tendem a ser demoradas e tediosas, pois são tarefas inevitáveis que não produzem novos conhecimentos. No entanto, a tarefa de reformatação de dados e de limpeza pode proporcionar ideias sobre quais suposições são seguras de serem feitas sobre os dados, quais peculiaridades existentes no processo de coleta e quais os modelos e análises são apropriados para serem aplicados.

A integração de dados é um desafio relacionado nesta fase, mas é discutido na Seção 2.8. Muitas vezes, a programação envolve trabalhar os dados em diferentes ferramentas, convertendo de um formato de dados para outro, extraíndo dados numéricos a partir de um texto, e administrando experimentos numéricos que envolvem um grande número de arquivos de dados e diretórios. Tais tarefas são muito mais rápidas para serem realizadas em uma linguagem como Python do que em Java ou C++, discutimos mais sobre isso na Seção 2.7.10.

2.4.3. Modelagem de Dados

Os dados gerados em espaços urbanos são geralmente associados com uma propriedade espacial ou espaço-temporais. Por exemplo, a localização de estabelecimentos são dados espaciais; dados meteorológicos e consumo de energia são dados temporais (também chamados de séries temporais, ou *stream*). Já os dados de RSPs possuem propriedades espaço-temporais simultaneamente.

Existem várias formatos de dados para modelar dados de RSPs, sendo bastante po-

¹¹Um *like* é uma interação do usuário com o Facebook em que ele demonstra que gostou de um item compartilhado.

¹²<http://ohmage.org>.

pular o uso de grafos [Zheng et al. 2014a]. Na Figura 2.2 ilustramos a criação de um grafo dirigido em que os nós representam os locais onde os dados foram compartilhados e com arestas que conectam localidades que foram compartilhadas pelo mesmo usuário. Usando este grafo podemos extrair diversas informações. De fato, a descoberta de conhecimento em dados de RSPs caminha junto com uma vasta gama de estudos que utilizam a teoria de grafos [Newman 2010, Newman 2003, Easley and Kleinberg 2010]. Como mostramos na Seção 2.6, técnicas bem conhecidas utilizadas para análise de grafos podem ser aplicadas diretamente para estudar grafos derivados de RSPs que refletem condições das cidades.

Alguns dos desafios sobre questões da dinâmica temporal relacionados com dados urbanos das RSPs são discutidos na Seção 2.8.

2.5. Análise de Dados Urbanos Provenientes de RSPs

Como os dados urbanos provenientes de RSPs podem ser muito complexos, um passo fundamental em qualquer investigação é caracterizar os dados coletados, a fim de entender suas limitações e utilidade. Com isso, nesta seção vamos estudar as propriedades de três RSPs para compartilhamento de localização (Foursquare, Gowalla e Brightkite¹³); uma RSP para compartilhamento de fotos (Instagram); bem como uma RSP para compartilhamento de alerta de trânsito (Waze).

2.5.1. Descrição dos Dados

Nesta subseção apresentaremos todos os *datasets* aqui considerados. Todos os dados foram coletados através do Twitter, que é um serviço de *microblogging*, ou seja, ele permite que os seus usuários enviem e recebam atualizações pessoais de outros contatos em textos de até 140 caracteres, conhecidos como “*tweets*”. Além de *tweets* de texto simples, os usuários também podem anunciar dados a partir de uma integração com outros serviços, como o Instagram, Foursquare e Waze. Neste caso, fotos do Instagram, *check-ins* do Foursquare ou alertas do Waze anunciadas no Twitter passam a ficar disponíveis publicamente, o que por padrão não acontece quando o dado é publicado unicamente nos sistemas analisados.

Alguns dos *datasets* que foram analisados: Foursquare1 (≈ 5 milhões de check-ins em abril de 2012 - 1 semana); Foursquare2 (≈ 12 milhões de check-ins entre fev2010-jan2011); Foursquare3 (≈ 4 milhões de check-ins em maio de 2013 - 2 semanas; Gowalla (≈ 6 milhões de check-ins entre fev2009-out2010); Brightkite (≈ 4 milhões check-ins entre abr2008-out2010); Instagram1 (≈ 2 milhões de fotos entre jun2012-jul2012); Instagram2 (≈ 2 milhões de fotos em maio 2013 - 2 semanas); Waze (+212 mil alertas entre dez2012-jun2013). Como podemos ver, os dados refletem diferentes períodos. Além disso, os *datasets* incluem uma quantia bastante significativa de dados: mais de 30 milhões de registros considerando todas as fontes.

Cada dado sensoriado (foto, *check-in* ou alerta) é composto de coordenadas GPS (latitude e longitude), do horário do compartilhamento do dado e do ID do usuário compartilhador. O *dataset* Foursquare1 possui informações extras sobre o tipo de local: categoria (por exemplo, comida) e um identificador do local. Mais informações sobre os *datasets* e como eles foram obtidos podem ser encontradas em [Cheng et al. 2011, Silva et al. 2012,

¹³As RSPs para compartilhamento de localização Gowalla e Brightkite não estão mais em funcionamento.

Silva et al. 2013c, Silva et al. 2013d, Silva et al. 2013e].

2.5.2. Cobertura da Rede

Nesta seção, estudamos a cobertura das RSPs analisadas em diferentes granularidades espaciais, começando por todo o planeta, depois cidades e, por fim, áreas específicas de uma cidade. A primeira constatação ao analisar esses dados é que a cobertura é bastante abrangente e tem escala planetária [Cheng et al. 2011, Silva et al. 2013a, Silva et al. 2013e].

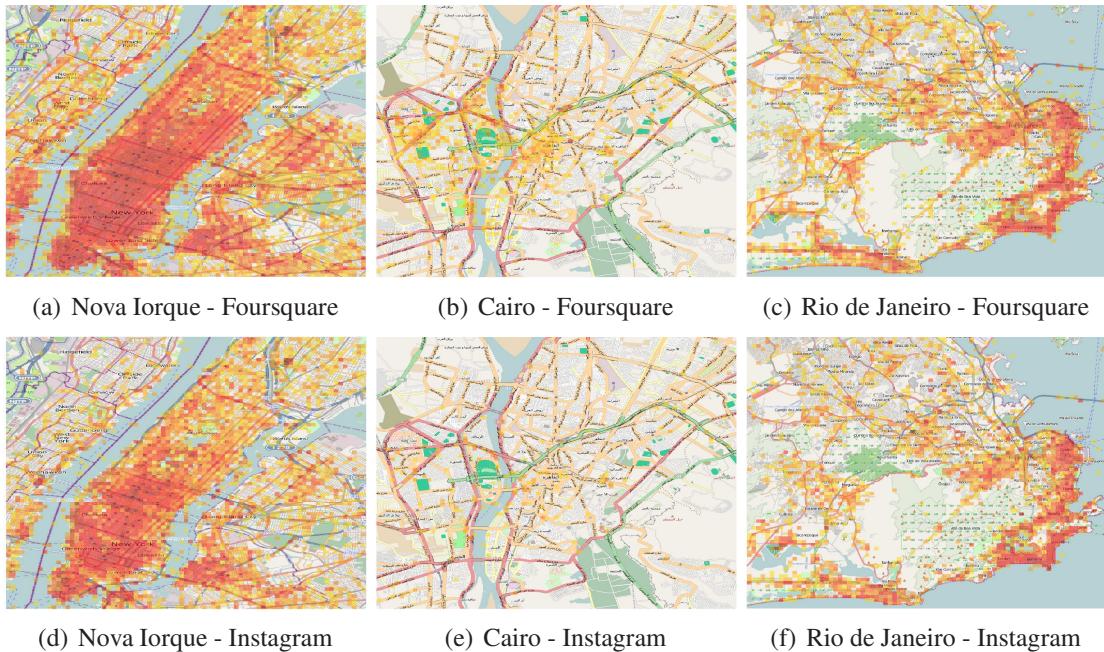


Figura 2.6. Cobertura espacial da RSP do Foursquare e Instagram em 3 cidades populosas ao redor do mundo [Silva et al. 2013a, Silva et al. 2013c].

Avaliamos agora a participação dos usuários em diversas cidades grandes, localizadas em regiões distintas, mostrando os resultados de algumas delas: Nova York, Rio de Janeiro e Cairo. A figura 2.6 mostra o mapa de calor da atividade de sensoriamento para cada uma dessas cidades. Mais uma vez, cores mais escuras representam um maior número de fotos em determinada área. Observamos uma alta cobertura para algumas cidades, como mostrado nas Figuras 2.6a e 2.6d (Nova York). No entanto, como podemos observar nas Figuras 2.6b e 2.6e, o sensoriamento no Cairo, que também possui um número elevado de habitantes, é significativamente mais baixo. Tamanha diferença na cobertura pode ser explicada por diversos fatores. Além dos aspectos econômicos, diferenças na cultura dos habitantes desta cidade, quando comparadas com as culturas presentes nas outras cidades estudadas, podem ter um impacto significativo na adoção e uso desses sistemas considerados [Barth 1969].

Além disso, pode-se observar que a cobertura em algumas cidades, como no Rio de Janeiro (Figuras 2.6c e 2.6f), é bem mais heterogênea quando comparada com a cobertura de Nova York. Isto ocorre, provavelmente, por causa dos aspectos geográficos particulares dessas cidades, ou seja, grandes áreas verdes e grandes porções d'água. O Rio de Janeiro tem a maior floresta urbana do mundo, localizada no meio da cidade, além de muitas colinas



Figura 2.7. Cobertura espacial da RSP para compartilhamento de alerta de trânsito no Rio de Janeiro [Silva et al. 2013e].

de difícil acesso humano. Estes aspectos geográficos limitam a cobertura do sensoriamento. Além disso, os pontos de interesse público, tais como pontos turísticos e centros comerciais, são distribuídos de forma desigual pela cidade. Há grandes áreas residenciais com poucos pontos desse tipo, enquanto outras áreas têm grande concentração dos mesmos.

A cobertura espacial dos dados da RSP para alertas de trânsito não é tão abrangente, como das RSPs para compartilhamento de localização e de foto. Isso pode ser observado na Figura 2.7, que mostra o número de alertas em diferentes regiões do Rio de Janeiro por um mapa de calor. Um fator que pode ajudar a explicar isso é a população de usuários do *dataset* de alertas de trânsito, que é menor do que nos outros casos estudados. Outro fator é que os usuários podem ter menos oportunidades para compartilhar alertas de trânsito em comparação com oportunidades para compartilhar fotos ou *check-ins*.

Como a atividade de participação pode ser bastante heterogênea dentro de uma cidade, analisamos a cobertura de RSPs em áreas específicas de uma cidade. Para ter um ID de uma área específica da cidade para os *datasets* do Instagram e Waze, propomos dividir a área das cidades em espaços retangulares menores, como em uma grade¹⁴. Chamaremos cada área retangular de uma *área específica* dentro de uma cidade. Consideraremos que uma área específica possui a seguinte delimitação: $1 \cdot 10^{-4}^\circ$ (latitude) $\times 1 \cdot 10^{-4}^\circ$ (longitude). Isso representa uma área de aproximadamente 8×11 metros em Nova Iorque e 10×11 metros no Rio de Janeiro. Para outras cidades, as áreas também podem variar um pouco, mas não a ponto de afetar significativamente as análises realizadas.

A Figura 2.8 apresenta a função de distribuição acumulada complementar (*complementary cumulative distribution function* - CCDF) do número de dados compartilhados (*check-ins*, fotos ou alertas) por área específica de todas as localidades em nossos *datasets*. Primeiramente, observe que, em ambos os casos, uma lei de potência¹⁵ descreve bem esta distribuição. Isso implica que, na maioria das áreas específicas, há poucos dados compartilhados, enquanto existem algumas poucas áreas com centenas de dados compartilhados. Estes resultados estão consistentes com os resultados apresentados em [Noulas et al. 2011a], trabalho que estudou a participação de usuários em sistemas de compartilhamento de localização. Nos sistemas analisados, é natural que algumas áreas possuam mais atividade que

¹⁴Note que nas áreas selecionadas não são consideradas fronteiras.

¹⁵Matematicamente, uma quantidade x segue uma lei de potência se ela pode ser obtida de uma distribuição de probabilidade $p(x) \propto x^{-\alpha}$, onde α é um parâmetro constante conhecido como expoente ou parâmetro escalar, e é um valor tipicamente entre $2 < \alpha < 3$ [Clauset et al. 2009].

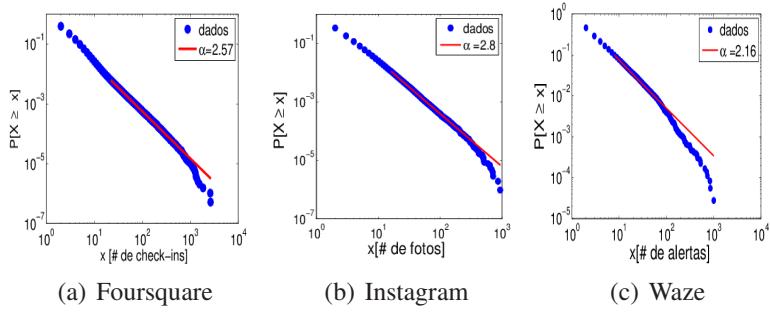


Figura 2.8. Distribuição do número de dados em áreas específicas (escala log-log) [Silva et al. 2013a, Silva et al. 2013a, Silva et al. 2013e].

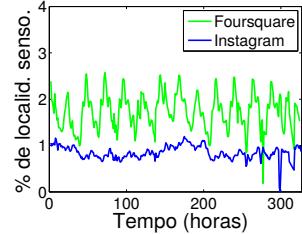


Figura 2.9. Porcentagem de áreas específicas sensoriadas ao longo do tempo [Silva et al. 2014a].

outras. Por exemplo, em áreas turísticas o número de fotos compartilhadas tende a ser maior do que em um supermercado, apesar de um supermercado ser geralmente um local bastante popular. Se uma determinada aplicação requer uma cobertura mais abrangente, é necessário incentivar os usuários a participarem em locais que eles usualmente não o fariam. Micro pagamentos ou sistemas de pontuação são exemplos de alternativas que poderiam funcionar nesse caso. Discutimos essas oportunidades na Seção 2.8.3.

Mostramos que uma RSP pode ter uma cobertura em escala planetária. No entanto, essa cobertura pode ser bastante desigual, em que grandes áreas ficam praticamente descobertas. Com isso em mente, a Figura 2.9 mostra a percentagem de locais distintos onde os usuários compartilharam dados em um determinado intervalo de tempo no Instagram e Foursquare¹⁶, que possuem 598.397 e 725.419 locais, respectivamente. O percentual máximo de locais distintos compartilhados por hora é inferior a 3% para todos os sistemas. Isto indica que a cobertura instantânea destas RSPs é muito limitada quando consideramos todas as localidades que poderiam ser sensoriadas no planeta (considerando todas as localidades já sensoriadas pelo menos uma vez). Em outras palavras, a probabilidade de uma área específica aleatória ser sensoriada em um horário aleatório é bem baixa.

2.5.3. Rotinas e o Compartilhamento de Dados

Analisamos agora como a rotina dos humanos afeta o compartilhamento dos dados. A Figura 2.10 mostra o padrão semanal de compartilhamento de dados em todos os tipos de RSPs analisadas¹⁷. Como esperado, os dados compartilhados nas RSPs apresentam um padrão diurno, o que implica que durante a madrugada a atividade de sensoriamento é bastante baixa.

Considerando dias de semana, é possível observar um ligeiro aumento da atividade ao longo da semana, com poucas exceções quando há um pico de atividade. O trabalho [Cheng et al. 2011], que analisou sistemas para compartilhamento de localização, foi observado esse mesmo comportamento, sem nenhum dia como exceção.

Podemos ainda observar que alguns picos de atividade variam ao longo do dia de

¹⁶Consideramos os datasets Instagram2 e Foursquare3, pois representam o mesmo intervalo de tempo.

¹⁷O horário do compartilhamento foi normalizado de acordo com o local onde o dado foi compartilhado, utilizando para isso a informação geográfica do local.

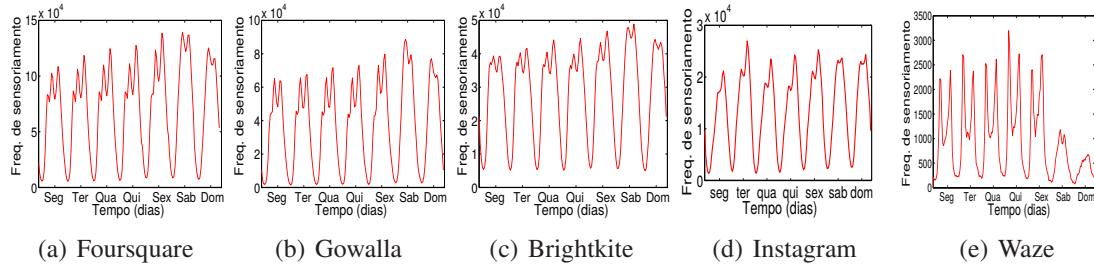


Figura 2.10. Padrão do compartilhamento de fotos durante os dias da semana [Silva et al. 2013a, Silva et al. 2013c, Silva et al. 2013e].

acordo com o propósito da RSP. Como podemos ver na Figura 2.10, na RSP para compartilhamento de localizações (Figuras 2.10a–c) existem três picos evidentes por volta da hora do café da manhã, almoço e jantar. Isso também foi observado em [Cheng et al. 2011]. Já na RSP para compartilhamento de fotos (Figura 2.10d) existem apenas dois picos evidentes, que ocorrem por volta da hora do almoço e jantar. E no caso da RSP para compartilhamento de alertas de trânsito (Figura 2.10e) também existem dois picos evidentes, um por volta de 7:00 e 8:00 da manhã e outro por volta de 6:00 da tarde, coincidindo com horários típicos de maior intensidade no trânsito.

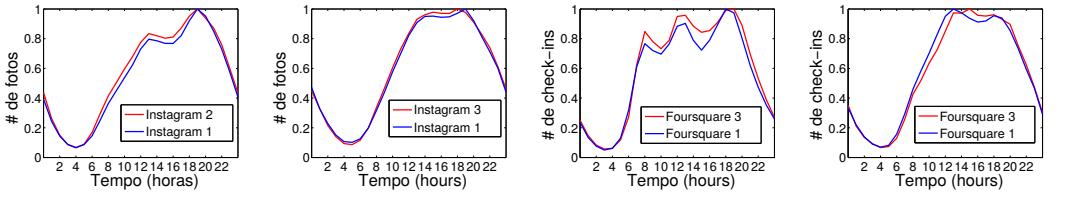
Analisando os diferentes padrões de comportamento para dias de semana e final de semana podemos observar que o padrão é significativamente diferente. Note que os picos observados nos dias de semana não são evidentes nos finais de semana. A falta de rotina bem definida nos fins de semana é uma das possíveis explicações para esse fato. Além disso, as diferenças entre dias de semana e final de semana possuem relação com o tipo de sistema analisado. Por exemplo, como nos fins de semana muitas pessoas não precisam dirigir, é natural esperar um volume menor de dados no Waze.

A Figura 2.11 mostra o padrão temporal de compartilhamento para o Instagram e o Foursquare considerando todos os *datasets*. Essa figura apresenta o número médio de dados compartilhados por hora durante, durante os dias de semana (de segunda a sexta-feira) e também durante o fim de semana (sábado e domingo). Surpreendentemente, vemos o mesmo padrão de compartilhamento para cada curva é muito semelhante, apesar do enorme intervalo entre as coletas (aproximadamente um ano). Isso acontece para os dias de semana e fins de semana, sugerindo que o comportamento do usuário em ambos os sistemas tende a se manter consistente ao longo do tempo. Esse é um resultado interessante e importante, pois mostra que podemos usar diferentes *datasets* para propósitos similares.

Mostramos agora como as rotinas impactam no comportamento de compartilhamento durante a semana. Para essa análise, consideramos os *datasets* do Instagram e Foursquare para Nova York, São Paulo e Tóquio. Os resultados são mostrados na Figura 2.12¹⁸. Em todas as figuras nós exibimos dados dos *datasets* do mesmo período (Instagram2 e Foursquare3) para duas cidades do mesmo país, e dados de um *dataset* com período anterior (Instagram1 e Foursquare1) para uma dessas cidades, como uma referência de comparação.

Primeiramente, observe a distinção entre as curvas de cada cidade no mesmo sis-

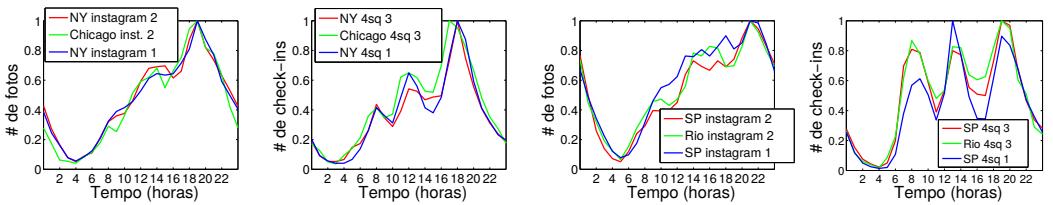
¹⁸Cada curva é normalizada pelo número máximo de conteúdo compartilhado em uma região específica representando a cidade.



(a) Instagram – dia de semana (b) Instagram – fim de semana (c) Foursquare – dia de semana (d) Foursquare – fim de semana

Figura 2.11. Padrão de compartilhamento temporal no Instagram e Foursquare [Silva et al. 2013d].

tema (por exemplo, Instagram, Figuras 2.12a, c, e) e também em diferentes sistemas (por exemplo, as Figuras 2.12a e 2.12b para Nova Iorque). Em seguida, observe que o padrão de compartilhamento para cada cidade no mesmo país é bastante semelhante, o que pode ser consequência dos padrões culturais dos habitantes desses países. Isso representa, de certa maneira, uma assinatura de aspectos culturais, o que ilustra, mais uma vez, o potencial desse tipo de dado para o estudo de dinâmica de cidades e do comportamento social urbano.



(a) Nova Iorque – Instagram (b) Nova Iorque – Foursquare (c) São Paulo – Instagram (d) São Paulo – Foursquare

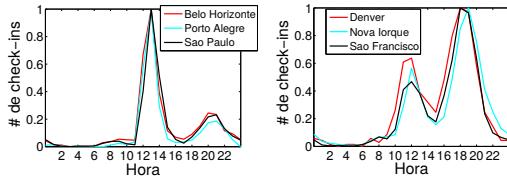
Figura 2.12. Padrão de compartilhamento temporal do Instagram e Foursquare para Nova Iorque, São Paulo e Tóquio durante dias de semana [Silva et al. 2013d].

Podemos ainda analisar classes de locais específicos. As Figuras 2.13a e 2.13b¹⁹ mostram o número de *check-ins* realizados em restaurantes ao longo das horas do dia, durante os dias de semana, em diferentes cidades do Brasil e dos Estados Unidos. Estes resultados capturaram diferenças importantes entre as culturas dos dois países: enquanto o jantar é a refeição principal para os americanos, o almoço desempenha um papel mais importante nos hábitos alimentares dos brasileiros.

2.5.4. Comportamento dos Nós

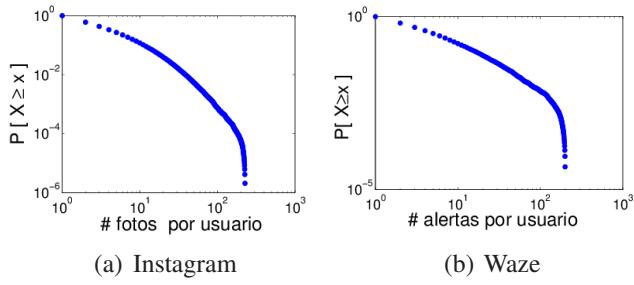
Nesta seção analisamos o desempenho dos nós da RSP (i.e., dos usuários) quanto ao compartilhamento de dados. A Figura 2.14 mostra a distribuição do número de dados (fotos e alertas) compartilhados por cada usuário da nossa base de dados. Como podemos observar, a distribuição possui cauda pesada, o que significa que a participação dos usuários pode ser muito desigual. Por exemplo, aproximadamente 40% dos usuários contribuíram com apenas uma foto no período considerado, enquanto que somente 17% e 0,1% dos usuários contribuíram com mais que 10 e 100 fotos, respectivamente. É natural que essa variabilidade

¹⁹Os valores são normalizados pelo valor máximo encontrado em qualquer hora para a cidade específica.



(a) Cidades brasileiras (b) Cidades dos EUA

Figura 2.13. Número médio de *check-ins* em restaurantes durante dias de semana ao longo das horas do dia [Silva et al. 2014a].



(a) Instagram

(b) Waze

Figura 2.14. Distribuição do número de dados compartilhadas pelos usuários [Silva et al. 2013c, Silva et al. 2013e].

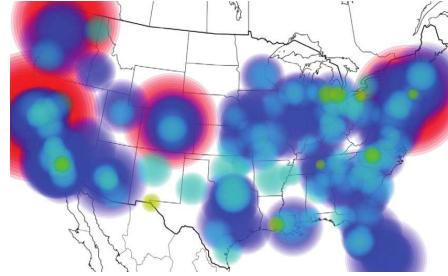


Figura 2.15. Resultado da métrica raio de giro [Cheng et al. 2011].

aconteça por diversos motivos. Por exemplo, alguns usuários podem dar mais importância para quesitos de privacidade do que outros. Uma cauda pesada também é observada na distribuição do número de *check-ins*, como foi mostrado em [Noulas et al. 2011a]. Cerca de 20% dos usuários realizaram apenas um *check-in*, 40 % acima de 10, ao passo que cerca de 10 % realizaram mais de 100 *check-ins*.

Além disso, em [Silva et al. 2013c, Silva et al. 2013e, Silva et al. 2013a] mostramos que há momentos em que muitos dados são compartilhados em intervalos de poucos minutos e momentos em que não há compartilhamento por horas. Isso pode indicar que a maioria do compartilhamento de dados acontece em intervalos específicos, provavelmente relacionados ao ciclo circadiano (ou rotina) das pessoas. Por exemplo, o compartilhamento de fotos em restaurantes tende a acontecer mais nos horários de almoço e jantar. Aplicações baseadas nesse tipo de sensoriamento devem considerar que a participação do usuário pode variar significativamente ao longo do tempo.

Observamos ainda que uma fatia significativa dos usuários realiza compartilhamento consecutivo de fotos em um curto intervalo de tempo. Por exemplo, cerca de 20% de todo o compartilhamento de fotos observado acontece em até 10 minutos. Isso sugere que os usuários tendem a compartilhar mais de uma foto na mesma área. Em [Noulas et al. 2011a] os autores também observaram que uma parcela significativa dos *check-ins* no Foursquare são realizados dentro de um curto intervalo de tempo. Por exemplo, mais do que 10% de *check-ins* ocorrem dentro de 10 minutos.

Em [Cheng et al. 2011] os autores analisaram *check-ins* compartilhados em vários serviços de compartilhamento de localização. Eles descobriram que os usuários possuem padrões simples e reproduzíveis, e também que o status social, além de fatores geográficos

e econômicos, colaboram com a mobilidade.

Para fazer essa análise os autores usaram três propriedades estatísticas para estudar e modelar padrões de mobilidade humana: *deslocamento (displacement)*; *raio de giro (radius of gyration)*; e *probabilidade de retorno (returning probability)*. Para ilustrar um de seus resultados, a Figura 2.15 mostra o raio médio de giro dos usuários em grandes cidades (com mais de 100.000 habitantes) nos EUA. As bolhas vermelhas²⁰ são cidades com um raio de giro maior do que 500 milhas; as azuis são cidades com um raio maior do que 250 milhas; as de cor ciano possuem um raio maior do que 125 milhas; e as amarelos são o resto das grandes cidades analisadas. Usuários em cidades costeiras tendem a ter um raio maior de giro do que os usuários em cidades do interior, e as pessoas em estados centrais tendem a ter um alto raio de giro devido a viagens de longa distância para o litoral [Cheng et al. 2011].

Da mesma forma, em [Cho et al. 2011] os autores investigaram padrões de movimentos e como os laços sociais podem impactar nesses movimentos. Os autores observaram que viagens de curta distância são espacialmente e temporalmente periódicas e não são afetadas pela estrutura de rede social, enquanto as viagens de longa distância são mais influenciadas por laços da rede social.

2.5.5. Considerações Finais

Identificamos várias propriedades de RSPs em comum: (i) possuem escala planetária; (ii) possuem uma frequência altamente desigual de compartilhamento de dados, tanto espacialmente quanto temporalmente, o que é altamente correlacionado com a rotina típica das pessoas; (iii) a participação do usuário em relação ao número de dados compartilhados e onde esses dados são compartilhados pode variar significativamente; (iv) o padrão temporal de compartilhamento parece não variar consideravelmente ao longo do tempo para o mesmo tipo de sistema.

As propriedades identificadas revelam o potencial de RSPs para conduzir vários estudos sobre a dinâmica da cidade e do comportamento social urbano. Além disso, o entendimento do comportamento do usuário é o primeiro passo para modelá-lo. Com modelos que explicam o comportamento do usuário podemos fazer previsões de ações e desenvolver melhores sistemas para planejamento de capacidade de carga do sistema.

Na Seção 2.7.10 são discutidas algumas das principais tecnologias e ferramentas para a análise de dados de RSPs, que podem ser bastante úteis em futuras análises de outros dados.

2.6. Aplicações e Serviços Relacionados ao Estudo de Sociedades Urbanas

Nesta seção, discutiremos as abordagens e modelos utilizados em diversas aplicações e serviços relacionados ao estudo da dinâmica da cidade e do comportamento social urbano utilizando dados de RSPs. Para a construção de novos serviços e aplicações nessa área é de suma importância conhecer as propriedades dos dados da RSP em estudo.

Os estudos mostrados aqui foram agrupados em cinco classes: Funcionamento de Cidades (Seção 2.6.1); Mobilidade Urbana (Seção 2.6.2); Padrões Sociais, Econômicos e Culturais (Seção 2.6.3); Detecção de Eventos e Interesses (Seção 2.6.4); e Problemas das

²⁰Consultar a versão digital disponível online em cores.

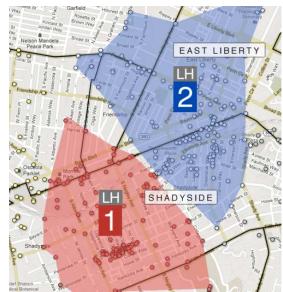


Figura 2.16. “Livehoods” encontrados em Nova Iorque [Cranshaw et al. 2012].

Cidades (Seção 2.6.5).

2.6.1. Funcionamento de Cidades

As informações obtidas a partir de RSPs têm o poder de mudar os nossos limites físicos percebidos, bem como ajudar a compreender melhor a dinâmica de cidades. Esta seção concentra na apresentação de estudos nessas direções.

Usando dados do Foursquare, em [Cranshaw et al. 2012] os autores propuseram um modelo para identificar regiões distintas de uma cidade que refletem padrões atuais de atividades coletivas, apresentando novos limites para os bairros. A ideia é expor a natureza dinâmica das áreas urbanas locais, considerando a proximidade espacial (derivado de coordenadas geográficas) e proximidade social (derivado da distribuição de check-ins) de locais.

Para isso, os autores utilizaram dados do Foursquare e desenvolveram um modelo que agrupa locais semelhantes considerando características sociais e espaciais. Cada *cluster* representa diferentes fronteiras geográficas dos bairros. O método de agrupamento utilizado é uma variação do agrupamento espectral proposto por [Ng et al. 2002].

A Figura 2.16 mostra dois *clusters* (ou “livehoods”, nome usado pelos autores), encontrados em Nova Iorque, representados pelos números 1 e 2. Nessa figura as linhas pretas indicam os limites oficiais da cidade. Veja que os limites dos *clusters* são bastante diferentes. Para tentar validar esses resultados os autores usaram resultados de entrevistas com moradores da cidade. De acordo com as respostas coletadas, esses e outros *clusters* eram esperados.

Em [Noulas et al. 2011b] os autores propuseram uma abordagem para classificar áreas e usuários de uma cidade usando categorias de locais do Foursquare. Isso poderia ser usado para identificar as comunidades de usuários que visitam categorias semelhantes de lugares, útil para sistemas de recomendação, ou na comparação de áreas urbanas dentro e entre as cidades. A abordagem utilizada é baseada em algoritmo de agrupamento espectral [Ng et al. 2002].

Mais especificamente, os autores consideram a atividade dos usuários do Foursquare para Nova Iorque, como mostrado pela Figura 2.17. Nessa figura, um círculo representa um local e seu raio a popularidade em termos de número de checkins. Cada cor corresponde a uma das oito categorias gerais introduzidas pelo Foursquare. Essa figura destaca a diversidade da atividade humana sobre a área considerada.

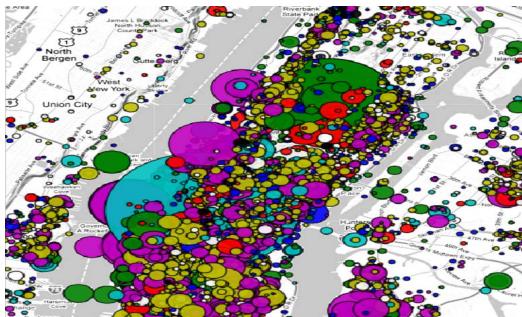


Figura 2.17. Atividade dos usuários do Foursquare para Nova Iorque. A&E (vermelho); Edu (preto); Outd (verde); NL (magenta); Shop (branco); and Trvl (ciano) [Noulas et al. 2011b].

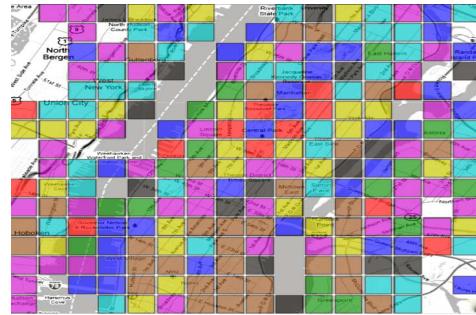


Figura 2.18. Visualização do agrupamento espectral. Cada cor simboliza um cluster. [Noulas et al. 2011b].

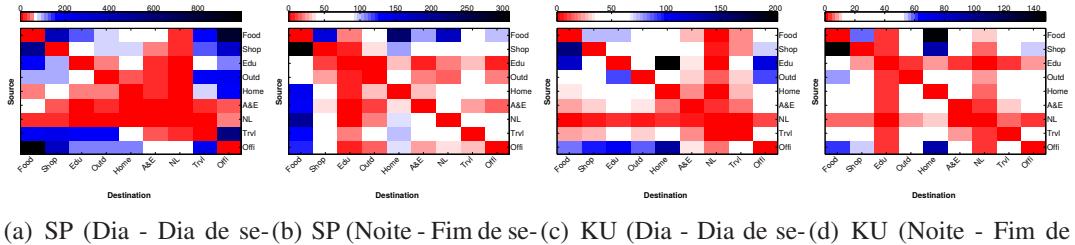
Em seguida, os autores dividiram uma cidade para ser analisada em áreas de tamanhos iguais, cada uma delas será um dado de entrada para o algoritmo de agrupamento. Para cada área é calculada a atividade realizada pelos usuários com base nas visitas em locais dessa região. Com isso, calcula-se a semelhança entre duas áreas utilizando a similaridade do cosseno (*cosine similarity*), entre as atividades representadas. Após esse processo, os autores realizam um agrupamento espectral. O resultado desse processo para a cidade de Nova Iorque é mostrado na Figura 2.18.

Em [Silva et al. 2014d] propusemos uma técnica chamada *City Image*, que fornece um resumo visual da dinâmica da cidade com base nos movimentos das pessoas. Esta técnica explora grafos de transição urbana para mapear os movimentos dos usuários entre locais da cidade. O grafo de transição urbana considerado é um grafo dirigido ponderado $G(V, E)$, em que um nó $v_i \in V$ é a categoria de um local específico (por exemplo, *food*) e uma aresta direcionada $(i, j) \in E$ marca uma transição entre duas categorias. Ou seja, uma aresta existe a partir do nó v_i para o nó v_j se pelo menos um usuário compartilhou um dado em um local categorizado por v_j logo após compartilhar um dado em um local categorizado por v_i . O peso $w(i, j)$ de uma aresta é o número total de transições que ocorreram a partir de v_i para v_j . Somente dados consecutivos compartilhados pelo mesmo usuário dentro de 24 horas, com início às 5:00, são considerados no cálculo de uma transição.

A *City Image* é uma técnica promissora que permite uma melhor compreensão da dinâmica de cidades, ajudando na visualização das rotinas comuns de seus cidadãos. Cada célula da *City Image* representa o quanto favorável é uma transição de uma determinada categoria em um determinado lugar (eixo vertical) para outra categoria (eixo horizontal), valores que são calculados utilizando um modelo aleatório/nulo [Silva et al. 2014d]. As cores vermelhas representam rejeição, as cores azuis representam favorabilidade e a cor branca representa indiferença. Nós exemplificamos a técnica *City Image* para duas cidades²¹: São Paulo (Figures 2.19a and 2.19b); e Kuwait (Figures 2.19c e 2.19d). Para ambos os casos, consideramos dias de semana durante o dia, que é o período típico de rotinas, e fim de semana durante a noite, que é um período representativo de atividades de lazer (fora da rotina).

Primeiramente, observe que transições para *office* (locais de trabalho) são mais pro-

²¹Utilizando dados do dataset Foursquare1.



(a) SP (Dia - Dia de semana) (b) SP (Noite - Fim de semana) (c) KU (Dia - Dia de semana) (d) KU (Noite - Fim de semana)

Figura 2.19. Images produzidas com a técnica *City Image* para São Paulo (SP) e Kuwait (KU) em diferentes períodos. Abreviaturas das categorias de locais (Nomes usados pelo Foursquare): Arts & Entertainment (A&E); College & Education (Edu); Great Outdoors (Outd); Nightlife Spot (NL); Shop & Service (Shop); and Travel Spot (Trvl) [Silva et al. 2014a].

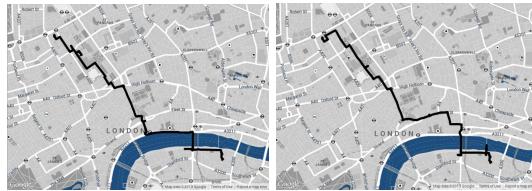
váveis de acontecer nos dias de semana e durante o dia para ambas as cidades, como esperado. No entanto, note que as imagens da cidade de São Paulo e Kuwait também têm diferenças significativas que refletem diversidades culturais entre ambas as cidades. Note, por exemplo, que a imagem que representa transições nas noites do fim de semana (Figure 2.19d) mostra a falta de transições favoráveis para a categoria *nightlife* no Kuwait. Este não é o caso de São Paulo (Figure 2.19b), em que a transição *food* → *nightlife* é altamente favorável de acontecer. Isso sugere que em São Paulo as pessoas gostam de frequentar locais relacionados com comida (*food*) antes de ir para casas noturnas (*nightlife*). No Kuwait, em vez disso, as pessoas são provavelmente mais favoráveis a realizarem as transições *shop* → *food* e *food* → *home* nas noites do fim de semana.

Técnicas para facilitar a interpretação das rotinas de habitantes de uma cidade, tais como as mencionadas aqui, são ferramentas valiosas para ajudar os urbanistas a entender melhor a dinâmica de cidades e, consequentemente, tomar decisões mais eficazes em relação à problemas das cidades, por exemplo.

2.6.2. Mobilidade Urbana

Nesta seção apresentamos trabalhos que se concentram em estudar padrões de mobilidade urbana dos usuários com dados de RSPs. Esses dados incluem informações espaço-temporais, por exemplo, *check-ins* e fotos com coordenadas geográficas. O estudo da mobilidade é útil para muitas finalidades. Com dados de RSPs é possível entender, por exemplo, como os usuários alocam tempo para diferentes atividades, sendo, portanto, uma questão fundamental nas ciências sociais. Além disso, é possível projetar novas ferramentas para ajudar os engenheiros de tráfego a entender o fluxo de pessoas na cidade.

A modelagem dos padrões de mobilidade vem atraindo a atenção de pesquisadores em diferentes áreas, como física e computação [Brockmann et al. 2006, Zheng et al. 2009, Gonzalez et al. 2008]. É importante ressaltar que os dados derivados das RSPs são diferentes de dados provenientes de *traces* de GPS ou de dados tradicionais do uso do telefone celular, como ligações telefônicas, e apresentam características especiais e variados contextos. Por exemplo, os *check-ins* em serviços de compartilhamento de localização ou fotos em um serviço de compartilhamento de fotos trazem informações extras sobre um lugar particular. Por exemplo, um *check-in* está associado com um tipo de local, e.g. bar, e uma foto



(a) Caminho mais curto (b) Caminho mais bonito

Figura 2.20. Mapas mostrando diferentes caminhos entre os mesmos locais [Quercia et al. 2014].

pode trazer informações sobre a situação atual dentro deste local. Com isso, nosso foco aqui são estudos que analisam dados de RSPs.

Em [Quercia et al. 2014] os autores propuseram uma metodologia para recomendação de rotas que leva em consideração não somente o menor caminho, mas também características emocionais, por exemplo, beleza. Nem sempre o menor caminho é o que gostaríamos de percorrer. Um turista, por exemplo, pode optar por um caminho mais bonito, mesmo que a distância seja um pouco maior. A Figura 2.20 mostra dois caminhos entre os mesmos locais na cidade de Londres, em que um é o mais curto (Figura 2.20a) e o outro o mais bonito (Figura 2.20b).

Para quantificar o quanto localidades urbanas são agradáveis, os autores usaram dados de uma plataforma de *crowd-sourcing* que mostra duas cenas de ruas em Londres (considerando centenas), e um usuário vota em qual acha mais bonita, tranquila e feliz. Em seguida, os autores traduzem os votos em medidas quantitativas de percepção de localização. Depois disso, os autores criam um grafo considerando essas localizações. Para isso, os autores dividiram a área da cidade em células de 200x200 metros. Cada célula é um nó no grafo e cada nó possui arestas com nós que representam células vizinhas. Esse grafo permite a descoberta de caminhos agradáveis.

Nguyen and Szymanski [Nguyen and Szymanski 2012] usaram dados do Gowalla para criar e validar modelos de mobilidade e relações humanas. Nesse trabalho, os autores propuseram um modelo de mobilidade baseado em amizade (FMM), que leva em conta os laços sociais, a fim de fornecer um modelo mais preciso da mobilidade humana. Com esse modelo, os autores foram capazes de estudar a frequência com que amigos viajam juntos. Ele pode melhorar a precisão de um número variado de aplicações, tais como engenharia de tráfego em redes de comunicação, sistemas de transporte e planejamento urbano.

O modelo de mobilidade proposto utiliza um modelo de Markov, onde os estados representam locais de *check-ins* e as ligações representam a probabilidade de ir de um lugar para outro. Por exemplo, a probabilidade de ir do trabalho para bar é definida como a razão entre o número de vezes que um determinado usuário executa um *check-in* em um bar logo após realizar um *check-in* no trabalho, e o número de vezes que o usuário realiza um *check-in* no trabalho.

Em [Zheng et al. 2012] os autores estudaram a mobilidade e padrões de viagem de turistas a partir de fotos compartilhadas no Flickr. A fim de extrair os padrões de viagem, os autores focaram as análises no movimento de turistas de acordo com as regiões atrativas e características topológicas de rotas de viagem feitas por diferentes turistas. Para isso,

primeiro é construído um banco de dados de caminhos turísticos com base no conceito de entropia de mobilidade (considerando entropia de Shannon [Shannon 1948]), usada para discriminar o movimento turístico do não turístico.

Em seguida, os autores propõem um método para descobrir regiões atrativas em uma cidade, usando para isso o algoritmo de agrupamento DBSCAN [Ester et al. 1996]. Para estudar o movimento turístico, os autores consideraram um modelo de Markov criado a partir da sequência de visitas nas regiões atrativas. Com isso, os autores podem estimar as estatísticas de visitantes que viajam de uma região para outra. Para estudar as características topológicas de rotas de turismo, os autores realizam um agrupamento de rotas de viagem, aplicando uma versão modificada da maior subsequência comum (*longest common subsequence*), como uma métrica de similaridade para minimizar o ruído.

Esses esforços ilustram o crescente interesse e o potencial de utilização de dados compartilhados em RSPs para estudar padrões de mobilidade de humanos em larga escala.

2.6.3. Padrões Sociais, Econômicos e Culturais

Os dados de RSPs também podem ser usados para estudar aspectos sociais, econômicos e culturais dos habitantes de cidades. Por exemplo, pode-se argumentar que uma pequena quantidade de dados compartilhados em uma área da cidade pode indicar uma falta de acesso à tecnologia por parte da população local, pois o uso de serviços de compartilhamento de localização muitas vezes dependem de *smartphones* e planos de dados 3G ou 4G que, geralmente, são caros. Nessa direção, em [Silva et al. 2013a] nós mostramos que a análise de dados de RSPs permitem a visualização de fatos interessantes relacionados com questões socioeconômicas de uma cidade. Por exemplo, dados de uma RSP para compartilhamento de localização para a cidade do Rio de Janeiro são escassos em áreas pobres, incluindo as que são localizadas muito perto de áreas ricas. Essa informação pode ser útil para gerar melhores políticas públicas nessas áreas. Note que a mesma informação pode ser obtida utilizando métodos tradicionais, tais como questionários, mas esse processo é muito mais lento e caro.

Com o intuito de melhor entender padrões sociais a partir da análise de dados de RSPs, em [Quercia et al. 2012] os autores estudaram como comunidades virtuais, observadas nos sistemas analisados, se assemelham às comunidades da vida real. Os autores testaram se teorias sociológicas estabelecidas de redes sociais da vida real são válidas nessas comunidades virtuais. Eles descobriram, por exemplo, que os influentes (*social brokers*) no Twitter são líderes de opinião que se arriscam “twittando” sobre diferentes temas. Eles também descobriram que a maioria dos usuários têm redes geograficamente locais, e que os influentes expressam não apenas emoções positivas, mas também negativas.

Para realizar este trabalho, os autores aplicaram métricas de rede que a literatura afirma que podem estar relacionadas com relações sociais, como a reciprocidade e restrição da rede [Quercia et al. 2012]. A reciprocidade r é a proporção de arestas em uma rede que são bidirecionais $L^{<->}$ em relação ao número total de arestas L : $r = \frac{L^{<->}}{L}$. Considerando uma rede social focada em um vértice (“ego”) e vértices e arestas a quem o ego está diretamente conectado, valores baixos de reciprocidade poderiam indicar, por exemplo, uma rede social de uma celebridade. A restrição da rede mede as oportunidades de se tornar influente (*brokerage opportunities*). Um alto valor de restrição da rede significa menos oportunidades.

Os autores usaram a formulação de Burt [Burt 1992] nesse caso específico. Mais detalhes sobre essas e outras métricas de rede que podem ser usadas para análises de relações sociais podem ser encontradas no livro: [Easley and Kleinberg 2010].

Em uma direção similar, em [Joseph et al. 2012] os autores analisaram um conjunto de dados do Foursquare para identificar grupos de pessoas e os lugares que elas visitam. O modelo utilizado foi capaz de identificar grupos de pessoas que representam grupos espacialmente próximos e pessoas que parecem ter interesses semelhantes.

A abordagem utilizada se baseia na ideia de modelos probabilísticos para tópicos. Para isso, eles usaram o modelo de alocação latente de Dirichlet (LDA - *Latent Dirichlet Allocation*) [Blei et al. 2003], que é, geralmente, utilizado para estudar documentos. Na instância do modelo, cada *check-in* de um usuário é encarado como uma palavra de um “documento” que representa um usuário, isso de forma análoga a documentos de texto, onde um documento pode ter muitas palavras.

Além disso, ao estudar o comportamento social de áreas específicas, uma das primeiras perguntas que surgem é: o quanto diferente uma cultura é de outra? Sabemos que os hábitos alimentares e de bebidas são capazes de descrever fortes diferenças culturais. Com base nisso, em [Silva et al. 2014c] propomos uma nova metodologia para a identificação de fronteiras culturais e semelhanças entre sociedades, considerando hábitos alimentares e de bebida. Para isso, foram usados *check-ins* do Foursquare para representar as preferências do usuário em relação ao que se come e bebe localmente, por exemplo, em uma determinada cidade.

Essa análise surpreendentemente diz muito sobre as diferenças e semelhanças entre as culturas. Para isso estudamos a correlação entre os *check-ins* dados em diferentes tipos de restaurantes para várias cidades ao redor do mundo. Observamos que as cidades de um mesmo país, onde os habitantes normalmente possuem cultura e hábitos alimentares semelhantes, têm as correlações mais fortes com relação às preferências de restaurante. Além de preferências para as categorias de alimentos, também podemos ver diferenças nos horários em que as pessoas vão a restaurantes e compartilham dados, como foi apresentado na Seção 2.5.3. Essas análises permitiram a proposição de uma metodologia para a identificação de culturas semelhantes, que pode ser aplicada em regiões de tamanhos variados, como países, cidades ou até mesmo bairros [Silva et al. 2014c]. Nessa metodologia é utilizado um algoritmo de agrupamento baseado em particionamento ($k - means$ [Hartigan and Wong 1979]), bem como a técnica de análise de componentes principais [Jolliffe 2002]. Os resultados para países e cidades são ilustrados nas Figuras 2.21a e 2.21b, mostrando como culturas semelhantes são bem separadas. Nessas figuras foram usados os dois principais componentes apenas para mostrar os resultados, no entanto a obtenção do resultado considerou todos os componentes.

As diferenças culturais utilizando dados de RSPs também foram estudadas em [Hochman and Schwartz 2012], que investigaram as preferências de cores em fotos compartilhadas no Instagram. Os autores encontraram diferenças consideráveis entre imagens de países com culturas distintas. Na mesma direção, em [Poblete et al. 2011] os autores investigaram como o comportamento de divulgação de conteúdo no Twitter varia entre alguns países, bem como as possíveis explicações para essas diferenças. A investigação das distinções culturais entre diferentes cidades e países é valiosa em muitas áreas e pode auxiliar

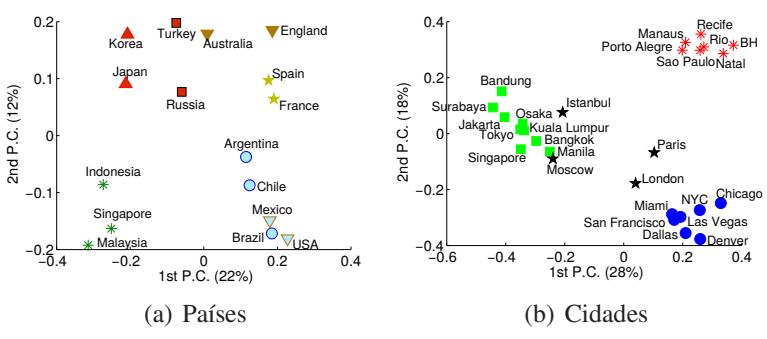


Figura 2.21. Grupos encontrados utilizando a metodologia de separação de culturas. Cada símbolo reflete um grupo [Silva et al. 2014c].



Figura 2.22. Localização das lojas analisadas [Karamshuk et al. 2013].

várias aplicações. Por exemplo, como cultura é um aspecto importante por razões econômicas, a identificação de semelhanças entre os lugares que estão geograficamente separados pode ser necessária para empresas que possuem negócios em um país e querem avaliar a compatibilidade de preferências entre diferentes mercados.

Relacionado com o aspecto econômico das cidades, em [Karamshuk et al. 2013] os autores estudaram o problema da alocação ótima de lojas de varejo na cidade. Eles usaram dados do Foursquare para compreender como a popularidade de três redes de lojas de varejo em Nova Iorque (veja a Figura 2.22) é definida em termos de número de *check-ins*.

Foram avaliadas um conjunto diversificado de características (*features*), modelando informações espaciais e semânticas sobre os locais e padrões de movimentos dos usuários na área ao redor do local analisado. Os autores observaram que a presença de locais que atraem muitos usuários naturalmente, como estação de trem ou aeroporto, bem como lojas de varejo do mesmo tipo das analisadas, que definem a concorrência comercial local de um área, são os indicadores mais fortes de popularidade.

2.6.4. Detecção de Eventos e Interesses

A identificação de eventos e pontos de interesse através de dados de RSPs é beneficiada pela natureza de tempo (quase) real das RSPs. Eventos podem ser naturais, tais como terremotos, ou não naturais, tais como a identificação/previsão de mudanças no mercado de ações. Por sua vez, um ponto de interesse é uma localização específica que alguém pode achar útil ou interessante, como um restaurante ou um estádio de futebol.

Em relação à detecção de eventos, em [Gomide et al. 2011] os autores analisaram como a epidemia de Dengue é refletida no Twitter e em que medida essa informação pode ser usada na vigilância dessa doença. Os autores mostraram que o Twitter pode ser usado para prever, espacial e temporalmente, epidemias de dengue. Eles analisam como dados do Twitter refletem epidemias com base em quatro dimensões: volume, localização, tempo e percepção do público. Especificamente, os autores estudam como os usuários se referem à dengue no Twitter com análise de sentimentos [Gonçalves et al. 2013] e usam o resultado para focar em apenas *tweets* que, de alguma forma, expressam experiência pessoal sobre a dengue. Em seguida, os autores construíram um modelo baseado em regressão linear [Yan 2009] para a previsão do número de casos de dengue. O resultado dessa pesquisa é

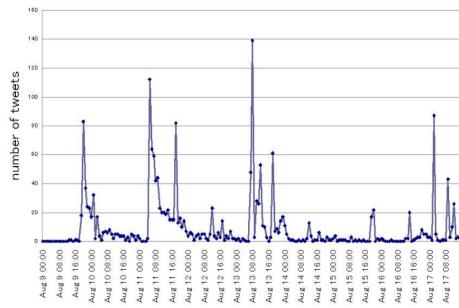


Figura 2.23. Número de tweets relacionados com terremotos [Sakaki et al. 2010b].

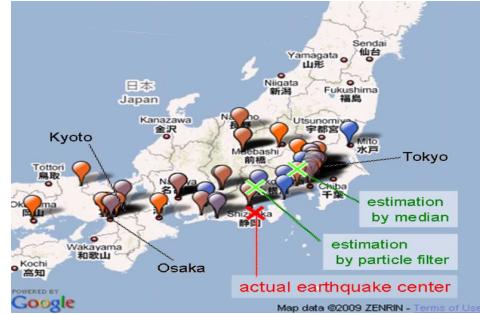


Figura 2.24. Estimativa da localização do terremoto [Sakaki et al. 2010b].

aplicado no projeto chamado Observatório da Dengue²².

Em [Sakaki et al. 2010b] os autores estudaram a interação em tempo real de acontecimentos no Twitter, por exemplo, terremotos, e propuseram um algoritmo para monitorar mensagens no Twitter para detectar a ocorrência de eventos. Para demonstrar a eficácia de seu método, os autores construíram um sistema de aviso de terremoto no Japão, que foi capaz de detectar 96% dos terremotos relatados pela Agência Meteorológica do Japão (JMA) com escala de intensidade sísmica de 3 ou mais. A notificação foi capaz de ser entregue mais rápida do que os avisos que são transmitidos pela JMA. A abordagem utilizada é um classificador de tweets com base em características, tais como: palavras-chave em um tweet; o número de palavras; e o seu contexto. Depois disso, os autores produziram um modelo espaço-temporal probabilístico para o evento alvo que pode encontrar o centro e a trajetória do local do evento.

Em os autores [Bollen et al. 2011] estudaram se os estados coletivos de humor derivados de mensagens do Twitter são correlacionados com o valor da bolsa Dow Jones ao longo do tempo. Seus resultados indicam que é possível obter uma boa precisão na previsão das mudanças diárias de alta e queda dos valores de fechamento dessa bolsa de valores. Isso é possível escolhendo dimensões de humor específicos, mas não todos os que foram considerados.

Além de eventos que tendem a acontecer esporadicamente, toda cidade possui um conjunto de áreas que desperta um maior interesse dos residentes ou visitantes, as aqui denominadas *pontos de interesse* (PDI). Dentre os PDIs mais visitados, podemos mencionar os pontos turísticos da cidade. No entanto, nem todos os PDIs de uma cidade são pontos turísticos. Por exemplo, uma área de bares pode ser bastante popular entre os residentes da cidade, mas sem atrativos para turistas. Além disso, PDIs são dinâmicos, ou seja, áreas que são populares hoje podem não ser mais amanhã. Assim, uma aplicação que emerge naturalmente a partir da análise de dados de algumas RSPs, por exemplo para compartilhamento de fotos ou localização, é a identificação de PDIs. Isso é possível porque cada foto ou check-in representa, implicitamente, um interesse de um indivíduo em um determinado instante. Com isso, quando muitas fotos de um determinado local são compartilhadas dentro de um certo intervalo de tempo, esse local pode ser um PDI.

²²<http://www.observatorio.inweb.org.br/dengue>.

Uma vantagem de usar RSPs para identificar pontos de interesse na cidade é que podemos obter resultados robustos a mudanças dinâmicas. Ou seja, pelo fato das RSPs fornecerem dados dinâmicos, elas podem capturar automaticamente as alterações nos interesses das pessoas ao longo do tempo, ajudando a identificar rapidamente as áreas que por ventura se tornem um PDI (por exemplo, devido à abertura de um novo restaurante) ou que deixem de ser populares.

A identificação de pontos de interesse em uma cidade foi investigada em [Crandall et al. 2009], onde os autores mostraram como inferir a localização de uma foto sem usar os dados geoespaciais. Na mesma direção, em [Kisilevich et al. 2010] os autores usaram fotos geolocalizadas para analisar e comparar eventos temporais que aconteceram em uma cidade, e também para classificar locais turísticos.

Além disso, em [Silva et al. 2013b] nós também apresentamos uma técnica para identificar PDIs e, a partir deles, identificar pontos turísticos. A técnica considera que cada par i de coordenadas (longitude, latitude) $(x, y)_i$ está associada a um ponto p_i , que representa um dado compartilhado, e.g. uma foto. Nós começamos calculando distância geográfica entre cada par de pontos (p_i, p_j) (usando a fórmula de Haversine [Sinnott 1984]) e agrupamos todos os pontos p_i que estão próximos uns dos outros (utilizando um método de agrupamento hierárquico aglomerativo, usando como critério de ligação: “complete-linkage” [Sørensen 1948, Kaufman and Rousseeuw 2009]). Para capturar os PDIs, usamos um modelo nulo para excluir grupos que possam ter sido gerados por situações aleatórias (ou seja, movimentos de pessoas aleatórias), e, portanto, não refletem a dinâmica da cidade. Para identificar os grupos, analisamos o número de compartilhamento de dados em cada um deles e usamos métodos estatísticos simples. Em seguida, separamos os pontos turísticos dos PDIs assumindo que turistas possuem rotas conhecidas na cidade (mais detalhes em [Silva et al. 2013b]).

Quando aplicada para a cidade de Belo Horizonte considerando dados do Foursquare e Instagram, essa técnica foi capaz de encontrar a maioria dos seus PDIs e pontos turísticos. Os resultados também mostram que diferentes RSPs podem fornecer dados complementares, pois nenhuma RSP encontrou todos os pontos turísticos. Tais diferenças podem refletir mudanças na cidade durante o intervalo de tempo em que um *dataset* específico foi coletado. Por exemplo, durante a coleta do *dataset* Instagram1, Belo Horizonte não estava recebendo jogos de futebol. Isso explica por que o estádio de futebol não foi identificado como um PDI utilizando esse *dataset*. Por outro lado, a análise de um *dataset* do mesmo sistema coletado mais recentemente (Instagram2), identificou corretamente o estádio como um ponto turístico importante da cidade. Isso ilustra como os dados de RSPs podem capturar automaticamente alterações da dinâmica da cidade, sendo úteis para detectar locais incomuns e populares, bem como descobrir possivelmente padrões inesperados.

2.6.5. Problemas das Cidades

A coleta de dados sobre problemas que as cidades enfrentam pode ser facilitada com o uso de RSPs como o Colab.re. Essa RSP permite aos usuários criar, visualizar e compartilhar problemas de diversas naturezas sobre a cidade. Além desse exemplo, existem outras RSPs para o monitoramento de questões específicas do meio ambiente urbano, como o nível de ruído. Por exemplo, NoiseTube [Maisonneuve et al. 2009], como apresentamos anteriormente.

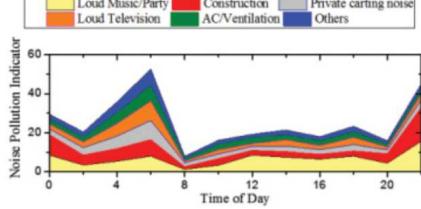
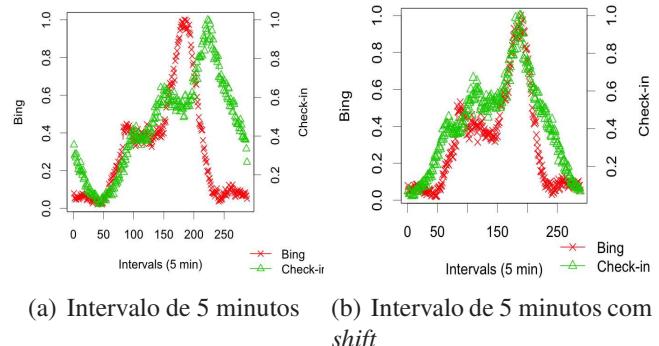


Figura 2.25. Ruídos de diferente categorias na Times Square [Zheng et al. 2014b].



(a) Intervalo de 5 minutos (b) Intervalo de 5 minutos com shift

Figura 2.26. Frequência de trânsito intenso em diferentes intervalos de tempo em um dia típico (segunda–sexta) [Tostes et al. 2014].

Com base no NoiseTube, D'Hondt e Stevens [D'Hondt et al. 2013] conduziram um experimento para mapear os níveis de ruído em Antwerp, Bélgica. Um dos objetivos era avaliar a qualidade dos mapas de ruído obtidos por sensoriamento participativo, em comparação com os mapas de ruído oficiais baseados em simulação. Para isso, foram realizados vários experimentos de calibração, investigando diversos aspectos dos padrões de ruído. Os autores foram capazes de construir mapas de ruído com uma margem de erro de 5dB, o que é comparado com mapas de ruído oficiais baseados em simulação.

Além dessas iniciativas, desde 2001, Nova Iorque disponibilizou uma plataforma chamada 311 para permitir que as pessoas reclamem de problemas da cidade usando um aplicativo móvel. Cada reclamação está associada a um local, data e hora, e, em alguns casos, informação detalhada da reclamação, como música alta ou barulho de construção (para os problemas de ruído). Usando os dados do serviço 311, em [Zheng et al. 2014b] os autores inferem a situação de ruído (consistindo de um indicador de poluição sonora), em diferentes momentos do dia para cada região de Nova Iorque. De acordo com o indicador de poluição sonora, é possível verificar a composição de ruído de um determinado local mudando ao longo do tempo (por exemplo, Time Square), como mostrado na Figura 2.25.

Os autores modelaram a situação de ruído de Nova Iorque com um tensor tridimensional, em que as três dimensões representam regiões, categorias de ruído e intervalos de tempo, o que permite recuperar a situação do ruído em toda a cidade. A informação de ruído não só pode facilitar a qualidade de vida de um indivíduo (por exemplo, ajudar a encontrar um lugar tranquilo para se estabelecer), mas também auxiliar os responsáveis governamentais no combate à poluição sonora.

Focados nos problemas de trânsito da cidade, em [Tostes et al. 2014] os autores estudaram a seguinte pergunta: é possível utilizar dados de RSPs como uma característica para predição de trânsito intenso? Os autores observaram que os dados de RSP, especialmente os provenientes do Instagram e Foursquare, são surpreendentemente bastante correlacionados com trânsito intenso e que podem ser utilizados para desenvolver modelos de previsão de congestionamento mais eficientes. Para tratar a questão, os autores desenvolveram uma Web *crawler* para a coleta de condições de tráfego do Google Maps e do Bing Maps. De posse desses dados, bem como com dados de RSPs, os autores estudaram a cidade de Nova

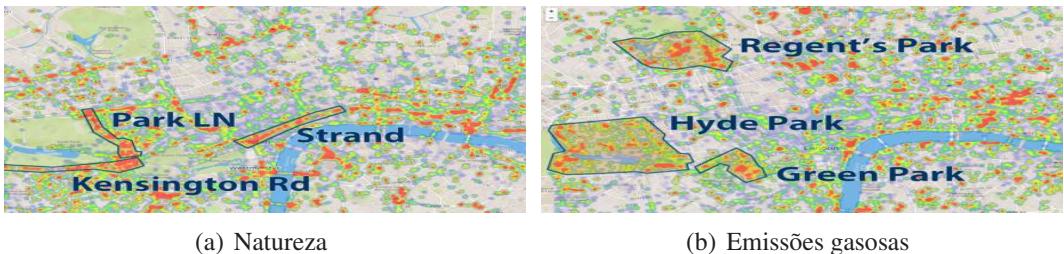


Figura 2.27. Mapas de cheiros em Londres [Quercia et al. 2015].

Iorque, mais especificamente a área de Manhattan. Conforme o estudo demonstra, os dados de RSP são bastante correlacionados com trânsito intenso, como mostrados na Figura 2.26. Ao comparar as Figuras 2.26a e 2.26b, podemos ver que a distribuição de trânsito intenso e a distribuição de *check-ins* durante os dias de semana são bastante semelhantes, no entanto as curvas são deslocadas no eixo x por um valor que pode ser calculado de acordo com uma equação proposta no estudo. Essa descoberta é surpreendente e sugere que dados de RSPs podem refletir as condições reais de tráfego. Em [Tostes et al. 2013] os autores propuseram também um modelo de previsão de tráfego utilizando uma regressão logística [Freedman 2009].

MacKerron and Mourato [MacKerron and Mourato 2013] estudaram como o ambiente local afeta a felicidade das pessoas. Para realizar esse estudo, os autores utilizaram uma aplicação para *smartphones* que tinha a finalidade de permitir aos usuários informarem o seu grau de humor, localização GPS e o nível de ruído do ambiente. Analisando mais de 3 milhões de registros de 45.000 pessoas no Reino Unido, os autores constataram que, em média, os participantes são significativamente mais felizes em ambientes externos, em contato com a natureza, do que em ambientes urbanos.

Em [Quercia et al. 2015] os autores exploraram a possibilidade de utilizar dados compartilhados em RSPs para mapear os cheiros percebidos em diversas regiões da cidade. Os resultados encontrados são promissores e mostram que essa pode ser uma nova forma para classificar áreas de acordo com o seu cheiro mais característico. Para realizar esse estudo, os autores consideraram dados do Instagram, Flickr e Twitter. Eles combinaram *tags* e *tweets* com as palavras de um “dicionário de cheiro” já existente. Em seguida, analisaram essas ocorrências na cidade. A Figura 2.27 mostra, utilizando um *heatmap*, a ocorrência de dados referentes a cheiros de natureza e emissão gasosa em Londres, Figuras 2.27a e 2.27b, respectivamente. É possível notar que o cheiro de natureza é fortemente observado em parques, e o cheiro de emissão gasosa em ruas com tráfego intenso.

2.6.6. Discussão

As RSPs oferecem informações atualizadas sobre locais, bem como opiniões e preferências de seus usuários. Além disso, elas têm o potencial de tratar as questões acima mencionadas em tempo (quase) real, atingindo um elevado número de regiões do globo. Nesta seção mostramos vários estudos que servem como exemplos de como trabalhar com dados de RSPs. As informações obtidas por esses estudos podem ser úteis para o desenvolvimento de serviços e aplicações mais inteligentes. Por exemplo, entender o padrão de comportamento em determinados locais na cidade, bem como a identificação de comportamentos fora do

padrão esperado pode ser muito útil para o planejamento de carga de uma rede celular urbana. Estudos que visam oferecer soluções para desafogar a transmissão de dados móveis (*mobile data offloading*) podem ter grandes benefícios ao utilizar essas informações como uma ferramenta para diminuir surpresas em demandas atuais, bem como novas demandas que podem surgir, já que a cidade está em constante mudanças. Várias outras oportunidades, bem como os desafios associados a elas, são discutidas na Seção 2.8.

2.7. Compilação de Técnicas e Ferramentas

Nesta seção discutimos as principais técnicas utilizadas nos trabalhos exemplificados nas seções anteriores, bem como algumas das tecnologias e ferramentas comumente utilizadas para a análise de dados urbanos. O objetivo não é fazer uma revisão completa da literatura. No entanto, acreditamos que os apontadores mostrados aqui podem ser bastante úteis no desenvolvimento de novas aplicações e serviços na área de computação urbana.

2.7.1. Algoritmos de Agrupamento

A realização de agrupamentos pode ser muito útil para a análise de dados urbanos, permitindo oferecer novos serviços por meio de agrupamento de características espaciais. A seguir, apresentaremos os principais algoritmos de agrupamento mencionados anteriormente.

Agrupamento baseado em densidade. Um exemplo de algoritmo popular dessa classe é o DBSCAN (*Density-based spatial clustering of applications with noise*) [Ester et al. 1996]. Algumas definições: a densidade é referente ao número de pontos dentro de um raio específico (ϵ); Um *core point* tem um número mínimo de pontos especificados pelo usuário ($minPts$) dentro do raio (ϵ); Um *border point* fica localizado na vizinhança de um *core point*; Um *noise point* é qualquer ponto que não se classifica como *core point* nem como *border point*.

A Figura 2.28 ilustra as definições mencionadas do DBSCAN, onde $minPts = 3$. O ponto *A* é um *core point*, pois pelo menos três pontos são vizinhos dele em um raio ϵ . Existem vários outros *core points* e como todos são todos acessíveis a partir de um outro *core point*, eles formam um único *cluster*. Os pontos *B* e *C* não são *core points*, mas são chamados de *border points*, porém são acessíveis a partir de *A* (que é um *core point*) e, portanto, também pertencem ao *cluster*. O ponto *N* é um *noise point*, pois não é um *core point* nem um *border point*.

Ideia do algoritmo: selecionar arbitrariamente um ponto p . Identificar todos os pontos densamente conectados a p com relação aos parâmetros ϵ e $minPts$. Se p é um *core point*, um *cluster* é formado. Se p é um *border point* e não há pontos densamente conectados a p o DBSCAN visita o próximo ponto do conjunto de dados. O processo continua até que todos os pontos tenham sido avaliados.

Agrupamento baseado em particionamento. Algoritmos de agrupamento desta classe constroem várias partições e as avalia usando algum critério. As partições são criadas a partir da segmentação de um conjunto de dados em um conjunto de k *clusters*. O objetivo é encontrar uma partição de k *clusters* que otimiza o critério de particionamento escolhido.

O $k - means$ [Hartigan and Wong 1979] é um algoritmo bastante popular para agru-

²⁴<https://en.wikipedia.org/wiki/File:DBSCAN-Illustration.svg>.

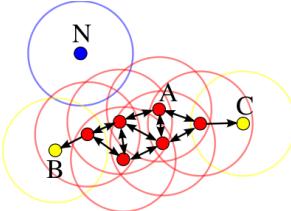


Figura 2.28. Diagrama ilustrativo do DBSCAN²⁴.

pamento baseado em particionamento. Esse algoritmo divide um conjunto de N amostras em k *clusters* disjuntos. Em linhas gerais, o algoritmo possui três passos. O primeiro é escolher os centroides iniciais, por exemplo, selecionando k amostras aleatórias do conjunto N . Após a inicialização, o $k - means$ considera um laço de repetição contendo dois outros passos: atribuir cada amostra a um centroide mais próximo; e criar novos centroides calculando o valor médio de todas as amostras designadas a cada centroide anterior. Essas duas etapas se repetem até que o centroide não se move significativamente.

Agrupamento hierárquico. O agrupamento hierárquico é um método que visa criar uma hierarquia de *clusters*. Estratégias de agrupamento hierárquico geralmente caem em dois tipos: (i) Aglomerativos: esta é uma abordagem “bottom up”, ou seja, cada observação começa em seu próprio *cluster*, e a cada passo combina-se *clusters* com alguma característica comum; e (ii) Divisivos: esta é uma abordagem “top down”, ou seja, todas as observações começam em um *cluster*, e divisões são realizadas de forma recursiva quando se desce na hierarquia, até que cada *cluster* tenha somente registros semelhantes.

A fim de decidir quais *clusters* devem ser combinados (por métodos aglomerativos), ou quando um *cluster* deve ser dividido (por métodos divisivos), é necessária uma medida de dissimilaridade entre os conjuntos de observações. Na maior parte dos métodos de agrupamento hierárquico, isto é alcançado pelo uso de uma medida apropriada de distância entre pares de observação e um critério de ligação (*linkage criteria*). Esse critério de ligação especifica a dissimilaridade dos conjuntos em função das distâncias entre pares de observações nos conjuntos. Um exemplo de critério de ligação é o *complete-linkage* [Sørensen 1948, Kaufman and Rousseeuw 2009]: $D_{AB} = \max \{ d(a, b) : a \in A, b \in B \}$, onde d é a métrica de distância escolhida, por exemplo, distância euclidiana, e A e B são conjuntos de observações. Por exemplo, valores de D_{AB} abaixo de um limite predefinido resultariam na aglomeração de A e B .

Agrupamento espectral. Em linhas gerais, o agrupamento espectral usa a similaridade entre os dados para definir os *clusters*. Para isso dois objetos matemáticos são usados: grafos de similaridade e grafos laplacianos (*graph Laplacians*). No grafo de similaridade cada vértice v_i do grafo representa um dado x_i do conjunto de dados, e dois vértices são conectados se a similaridade s_{ij} entre os dados correspondentes é positiva ou maior do que um certo limiar, com s_{ij} sendo o peso das arestas. A ideia é achar partições no grafo tal que arestas de diferentes *clusters* tenha peso baixo, enquanto que arestas dentro do mesmo *cluster* tenham pesos elevados [Luxburg 2007].

Os grafos laplacianos são as principais ferramentas para o agrupamento espectral. Diferentes algoritmos espectrais existem, cada um utiliza um grafo laplaciano descrito por

von Luxburg. A ideia principal por trás do algoritmo espectral é aumentar as propriedades de agrupamento dos dados em uma nova representação de maneira que os clusters sejam trivialmente detectados utilizando, por exemplo, algoritmos de como o k – *means*, como nos exemplos de [Luxburg 2007].

Outros métodos de agrupamento, bem como mais detalhes sobre as técnicas de agrupamento mencionadas aqui podem ser encontrados nos livros: [Zaki and Meira Jr 2014, Kaufman and Rousseeuw 2009].

2.7.2. Regressões

Em estatística, regressão é uma técnica que permite explorar e inferir a relação de uma variável dependente (variável de resposta) com variáveis independentes específicas (variáveis explanatórias). A regressão linear foi o primeiro tipo de regressão a ser estudado rigorosamente, bem como ser usado extensivamente em aplicações práticas, incluindo a predição de valores. A regressão linear é uma equação para se estimar a condicional (valor esperado) de uma variável y , dados os valores de algumas outras variáveis x . Com isso, em geral, a regressão linear trata da questão de se estimar um valor condicional não esperado [Yan 2009].

Para se estimar o valor esperado usa-se a equação: $Y_i = \alpha + \beta X_i + \varepsilon_i$, sendo que Y_i é a variável explicada; α é uma constante, que representa a interceptação da reta com o eixo vertical; β é outra constante, que representa o declive (coeficiente angular) da reta; X_i é uma variável explicativa (independente), que representa o fator explicativo na equação; ε_i é a variável que inclui todos os fatores residuais mais os possíveis erros de medição [Yan 2009].

Outra técnica utilizada nos trabalhos citados neste minicurso é a regressão logística [Freedman 2009]. A regressão logística é um tipo de regressão que tem como objetivo produzir, a partir de um conjunto de observações, um modelo que permita a predição de valores tomados por uma variável categórica a partir de uma série de variáveis explicativas contínuas e/ou binárias.

Em comparação com as técnicas conhecidas em regressão, em especial a regressão linear, a regressão logística distingue-se principalmente pelo fato da variável resposta ser categórica. Mais detalhes sobre outros modelos de regressão, bem como os que mencionamos aqui podem ser encontrados nos livros [Harrell 2013, Chatterjee and Hadi 2015, Freedman 2009].

2.7.3. Modelos Probabilísticos para Tópicos

Modelos probabilísticos para tópicos são usados, tipicamente, para entender e estruturar grandes coleções de documentos. O modelo de tópico mais comum é o modelo de alocação latente de Dirichlet (LDA – *Latent Dirichlet Allocation*), que utiliza a abordagem bayesiana para aprender a estrutura latente de temas que compreendem cada um dos documentos. Esse modelo vem sendo utilizado em muitos trabalhos de novas áreas, incluindo a computação urbana.

O LDA se baseia em um modelo generativo assumindo que cada um dos documentos em particular na coleção é uma mistura de temas. Num modelo LDA típico, um documento de texto é representado como um conjunto de palavras, onde cada palavra pertence a um ou mais tópicos ocultos. Assim, cada documento pode ser descrito ao considerar o quanto

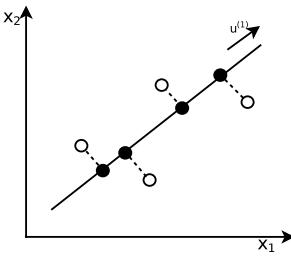


Figura 2.29. Exemplo de projeção ortogonal dos dados para uma dimensão menor: 2-D para 1-D.

as palavras dentro dele se relacionam com os vários tópicos ocultos e, cada tópico oculto pode ser descrito pelos termos que são mais fortemente associados a ele. Por exemplo, um documento sobre a abertura de um novo restaurante italiano pode conter as palavras “restaurante” e “jantar”, associadas ao Tópico 1, e as palavras “pizza” e “spaghetti”, associadas ao Tópico 2. Nesse caso, o LDA nos daria informações sobre a forma como o documento é relacionado aos dois tópicos e, com isso, poderíamos entender sobre o que esses dois tópicos dizem respeito considerando as palavras que estão associadas a eles. Por exemplo, o Tópico 2 é provável que seja sobre “alimentos” ou “comida italiana”.

Uma característica bastante interessante do modelo LDA é o mínimo de intervenção humana requerida para sua aplicação. O modelo LDA é capaz de descobrir tópicos relacionados a documentos e estabelecer links entre documentos, mesmo quando não temos nenhuma informação anterior sobre estes tópicos. Além disso, os documentos não são obrigados a ser rotulados com tópicos ou palavras-chave na inicialização do modelo LDA.

2.7.4. Redução de Dimensionalidade

O objetivo dessa classe de técnicas é sumarizar os dados que contêm muitas variáveis por um conjunto menor de variáveis compostas derivadas a partir do conjunto original. Uma das motivações para o uso destas técnicas é a compressão dos dados, o que permite economizar espaço em disco e memória, bem como aumentar o desempenho de processos de aprendizado de máquina. Outra motivação é proporcionar uma maneira melhor de visualizar resultados que possuem um grande número de características (dimensões).

A técnica análise de componentes principais (*principal components analysis - PCA* [Jolliffe 2002]), é uma técnica estatística amplamente utilizada para a redução de dimensão não supervisionada. O objetivo é encontrar k vectores $u^{(1)}, u^{(2)}, \dots, u^{(K)} \in \mathbb{R}^n$ para projetar os pontos de dados de modo a minimizar o erro de projeção. Em outras palavras, dado um conjunto de pontos de dados em um espaço dimensional, o objetivo é projetá-los em um espaço de dimensão menor, preservando o máximo possível de informações. A Figura 2.29 ilustra uma projeção ortogonal dos dados para uma dimensão menor: 2-D para 1-D. As linhas pontilhadas entre os pontos originais (fundo branco) e os pontos projetados na linha sólida (fundo preto) representam os erros de projeção que devem ser minimizados. Podemos utilizar o PCA, por exemplo, para encontrar a melhor aproximação planar de dados três dimensões, ou a melhor aproximação de doze dimensões dos dados com 10^4 dimensões.

Implementações para PCA podem ser encontradas em várias ferramentas. No R²⁵, as funções *princomp*²⁶ e *prcomp*²⁷ podem ser usadas para PCA. Em Matlab²⁸/Octave²⁹ a função *princomp*³⁰ calcula os componentes principais.

2.7.5. Análise de Sentimento

A análise de sentimento (também conhecida como mineração de opinião) refere-se ao uso de processamento de linguagem natural, análise de texto e linguística computacional para identificar e extrair informações subjetivas de materiais textuais. De um modo geral, a análise de sentimento tem como objetivo determinar a atitude de um autor com relação a algum tema ou a polaridade contextual global de um documento. A atitude pode ser o julgamento ou avaliação do autor, o seu estado afetivo (ou seja, o estado emocional quando estava escrevendo), ou a sua comunicação emocional pretendida (ou seja, o efeito emocional que se deseja ter no leitor) [Reis et al. 2015, Gonçalves et al. 2013].

A tarefa básica na análise de sentimento é classificar a polaridade da opinião expressa pelo autor em positiva, negativa ou neutra. Existem vários métodos que permitem a análise de sentimentos. No entanto, o método SentiStrength [Thelwall et al. 2010] é o exemplo aqui ilustrado, já que é bastante utilizado, possui boas taxas de precisão e foi desenvolvido para textos curtos (tamanho comumente encontrado em redes de sensoriamento participativo) [Gonçalves et al. 2013].

O SentiStrength atribui um sentimento positivo de força 1 (nenhum sentimento positivo) a 5 (sentimento positivo muito forte) e um sentimento negativo de força -1 (nenhum sentimento negativo) a -5 (sentimento negativo muito forte) para cada texto. Uma parte principal do SentiStrength é uma lista de várias palavras associadas a uma determinada força de sentimento positivo ou negativo. Por exemplo, “bom” possui valor 3 e “medo” possui valor -4, assim a frase “Eu estava com medo, mas foi bom” pode resultar 3 na escala positiva e -4 na escala negativa. Além disso, existem regras especiais para lidar com negações, perguntas, palavras de reforço (por exemplo, muito), *emoticons*, e uma série de outros casos especiais. Se for necessário um único número global para a força sentimento então o número positivo pode ser adicionado ao número negativo para dar uma pontuação no intervalo de -4 a 4 [Thelwall et al. 2010].

2.7.6. Teoria de Grafos/Redes

Estruturas que podem ser representadas por grafos/redes estão em toda parte e muitos problemas de interesse prático podem ser formulados como questões sobre certos grafos [Newman 2010]. Os “grafos de transição urbana” ($G(V, E)$) representam um exemplo de grafo bastante informativo sobre a dinâmica da cidade e do comportamento social urbano. Esse tipo de particular de grafo representa, por exemplo, um conjunto V de locais na cidade (denominados vértices) e um conjunto E de pares não ordenados de V que representam a movimentação dos usuários na cidade (as chamadas arestas).

²⁵<http://www.r-project.org>.

²⁶<http://stat.ethz.ch/R-manual/R-patched/library/stats/html/princomp.html>.

²⁷<http://stat.ethz.ch/R-manual/R-patched/library/stats/html/prcomp.html>.

²⁸<http://www.mathworks.com/products/matlab>.

²⁹<http://www.gnu.org/software/octave>.

³⁰<http://octave.sourceforge.net/statistics/function/princomp.html>.

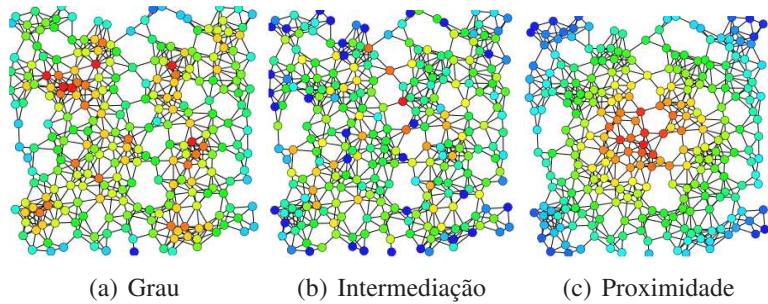


Figura 2.30. Resultados de centralidade considerando várias métricas para o mesmo grafo³².

A teoria de grafos é bastante rica, oferecendo várias métricas e conceitos, como métricas de centralidade. Em particular, essas métricas determinam a importância relativa de um vértice ou aresta no grafo, que no contexto de grafos de transição urbana representam o quanto um determinado local é importante. Algumas das medidas de centralidade que são amplamente utilizadas na análise de grafos: centralidade de grau (*degree centrality*), centralidade de intermediação (*betweenness centrality*) e centralidade de proximidade (*closeness centrality*) [Newman 2010].

A centralidade de grau é definida como o número de ligações incidentes sobre um vértice, ou seja, seu grau. Em um grafo de transição urbana, um vértice com grau elevado indica um local onde as pessoas podem chegar e sair com uma alta probabilidade. Assim, a centralidade de grau é uma boa medida para identificar lugares populares da cidade. Estes locais podem ser vistos como “*hubs*” da cidade.

A centralidade de intermediação mede a importância de um vértice dentro de um grafo (existe também a intermediação das arestas, que não é discutida aqui). A centralidade de intermediação quantifica o número de vezes que um vértice age como ponte ao longo do caminho mais curto entre dois outros vértices s e t . A intermediação de um vértice v pode ser calculada assim: $C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$, onde, σ_{st} é o número total de caminhos curtos desde o vértice s ao vértice t e $\sigma_{st}(v)$ é o número desses caminhos que passam por v . Essa métrica pode ser útil, por exemplo, para indicar os locais mais interessantes para atuarem como pontes para disseminar informações entre diferentes lugares ou regiões de lugares (conjunto de lugares) na cidade.

A centralidade de proximidade está relacionada com a proximidade de um vértice para todos os outros vértices na rede, ou seja, o número de arestas que separam um vértice a partir dos outros. No contexto de divulgação de informações, quanto maior a proximidade de um local, maior é a probabilidade de que a informação a ser disseminada a partir desse local alcance toda a rede na menor quantidade de tempo. Na perspectiva de um grafo de transição urbana, a centralidade de proximidade pode indicar locais estratégicos para instalar centros públicos para anúncio de informações à população. Uma ilustração do resultado de todas as métricas mencionadas aplicadas a um mesmo grafo é mostrada na Figura 2.30.

Existem várias outras métricas e conceitos de teoria de grafos além dos que foram

³²<https://upload.wikimedia.org/wikipedia/commons/thumb/1/14/Centrality.svg/600px-Centrality.svg.png>.

mencionadas no texto do minicurso. Uma revisão detalhada das principais métricas e conceitos pode ser encontrada em: [Newman 2010, Newman 2003, Easley and Kleinberg 2010]

2.7.7. Validações

Dados de RSPs podem possuir várias limitações. Em primeiro lugar, eles podem refletir o comportamento dos cidadãos que utilizam as RSPs: usuários que tendem a ter menos de 50 anos, donos de *smartphones* e moradores urbanos [Brenner and Smith 2013, Duggan and Smith 2014]. Além disso, os usuários podem não compartilhar dados em todos os seus destinos, por exemplo, hospitais e motéis. Em segundo lugar, os dados podem ser baseados em uma amostra limitada de dados. Isso significa que podemos ter apenas uma amostra das atividades realizadas. Fatores externos, tais como más condições meteorológicas, podem afetar o número total de dados que são coletados para alguns lugares, especialmente locais ao ar livre. Por isso, antes de tirar conclusões com dados de RSPs, é necessário realizar uma comparação com dados obtidos de maneira tradicional (*offline*). Existem várias fontes de dados disponíveis na Web que permitem realizar validações, algumas delas foram ilustradas em [Barbosa et al. 2014].

Para investigar a precisão do método para a identificação das fronteiras culturais [Silva et al. 2014c], os autores compararam os resultados obtidos com os resultados do World Values Surveys (considerado o mais importante da área disponível publicamente), que utilizou dados coletados de forma tradicional: questionários. Além disso, em [Silva et al. 2013c] os autores utilizaram dados do TripAdvisor (um popular website de turismo), para verificar se os resultados dos pontos turísticos encontrados eram relevantes na cidade estudada. Outro exemplo pode ser encontrado em [Cranshaw et al. 2012], onde os autores entrevistaram moradores da cidade estudada para validar os resultados obtidos.

2.7.8. Tratamentos Estatísticos

No estudo de propriedades de grafos, um modelo nulo (*null model*) é um grafo que corresponde a um grafo específico em algumas das suas propriedades estruturais, mas que também é considerado um exemplo de um grafo aleatório. O modelo nulo é usado, geralmente, como um termo de comparação para verificar se o grafo em questão apresenta alguma característica, como a estrutura de uma comunidade, ou não. Nem todos os modelos nulos são iguais. A criação de cada um depende de quais propriedades serão preservadas no grafo original e, também, qual é a hipótese nula sendo feita. Nas técnicas City Image e identificação de POIs, mencionadas anteriormente, foram utilizados modelos nulos que mantinham os vértices de grafos de transição urbana e distribuíam o mesmo número de arestas originais de forma aleatória (procedimento ilustrado na Figura 2.31, onde a grossura da aresta representa o seu peso). Esses modelos serviram para identificar transições que poderiam ser geradas de forma aleatória, bem como identificar transições populares. Além disso, este procedimento é importante porque pode nos impedir de usar dados que não têm relação com os fenômenos que estamos interessados.

Outro procedimento necessário para comparar propriedades de diferentes regiões (por exemplo, cidades), é realizar uma normalização dos dados [Freedman 2009]. Além disso, também é interessante verificar a razão de dados por habitantes, valor que, idealmente, deve ser similar para todas as regiões estudadas. Existem várias outras técnicas para

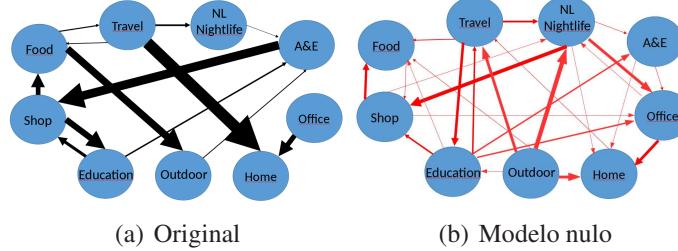


Figura 2.31. Grafo de transição urbana original e um modelo nulo.

realizar validações e tratamento estatístico de dados de RSPs para estudar o funcionamento de cidades e o comportamento urbano. A principal mensagem aqui é manter em mente que esses procedimentos são extremamente necessários e devem ser usados sempre que possível.

2.7.9. Aterfatos para o Estudo de Mobilidade

Nesta seção apresentamos mais detalhes sobre algumas das principais técnicas que foram utilizadas para o estudo de mobilidade em trabalhos citados anteriores.

Raio de Giro. O raio de giro mede a frequência e o quanto longe um usuário se move. Um baixo raio de giro normalmente indica um usuário que viaja principalmente localmente (com poucos compartilhamento de dados de longa distância), enquanto um alto raio de giro indica um usuário com muitos compartilhamento de dados de longa distância [Cheng et al. 2011]. O raio de giro de um usuário pode ser formalizado como:

$$r_g = \sqrt{\frac{1}{n} \sum_{i=1}^n (\mathbf{r}_i - \mathbf{r}_{cm})^2},$$

onde n é o número de dados do usuário, e $\mathbf{r}_i - \mathbf{r}_{cm}$ é a distância entre um determinado dado \mathbf{r}_i e centro de massa do usuário \mathbf{r}_{cm} (que é uma média simples da localização de todos os dados).

Modelo de gravidade. Na sua formulação mais simples, o modelo de gravidade (*Gravity Model* [Zipf 1946]) afirma que a interação $T_{i,j}$ entre dois locais i e j é proporcional ao produto das suas populações P_i e P_j sobre a sua distância $d_{i,j}$:

$$T_{i,j} = k \frac{P_i^\alpha P_j^\beta}{d_{i,j}^\gamma},$$

onde os expoentes α , β , γ e o fator de escala k são parâmetros ajustáveis, praticamente escolhidos de modo a ajustar os dados empíricos que estão sendo modelados. Ao longo dos anos, este modelo simples foi ampliado e refinado em uma variedade de maneiras, e sua aplicação tem sido além do domínio de transportes, o que mostra o seu potencial. Por exemplo, ele foi usado para modelar telefonemas interurbanos [Krings et al. 2009] e a propagação de doenças infecciosas [Balcan et al. 2009].

Fórmula de Haversine. A fórmula de Haversine é uma importante equação usada em navegação, fornecendo distâncias entre dois pontos de uma esfera a partir de suas latitudes e longitudes, podendo ser calculada como:

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right),$$

onde: d é a distância entre os dois pontos (ao longo de um grande círculo, *great circle*, da esfera); r é o raio da esfera (raio da Terra = 6372,8 km); ϕ_1, ϕ_2 : latitude do ponto 1 e latitude do ponto 2 e λ_1, λ_2 : longitude do ponto 1 e longitude do ponto 2.

Existem várias outras técnicas e modelos que auxiliam o estudo de mobilidade urbana utilizando dados de RSPs, além dos trabalhos citados, mais informações podem ser encontradas em outras áreas que também realizam estudos neste campo, como geografia, planejamento urbano e física.

2.7.10. Tecnologias e Ferramentas para a Análise de Dados

A análise dos dados urbanos é fundamental, pois para utilizar esses dados é necessário conhecer suas propriedades. O objetivo dessa seção é apresentar algumas das tecnologias e ferramentas comumente utilizadas para essa finalidade.

Quando se trata de análise de dados, a linguagem Python tanto quanto as linguagens Matlab/Octave ou R são relativamente fáceis para começar a codificar. A seguir mostramos algumas das vantagens de cada uma destas ferramentas.

Quando usar as linguagens Matlab ou R: Matlab e R possuem uma longa e confiável história, uma comunidade forte e com uso na indústria. Isto significa que se pode contar com o apoio online de outros usuários, caso precisar de ajuda ou tenha dúvidas sobre a utilização da linguagem. Além disso, há uma grande variedade de pacotes extras disponíveis publicamente. Quando é necessário fazer uma análise estatística pesada ou gráficos, Matlab ou R são as melhores opções. Operações matemáticos típicas como multiplicação de matrizes funcionam de forma bem simples e direta, bem como a visualização de dados.

Quando usar a linguagem Python: Se é necessário realizar uma limpeza e formatação dos dados, ou obter dados de *websites*, arquivos ou outras fontes de dados, a melhor escolha é Python. Em geral, devido à facilidade de transformar ideias em código e pela simplicidade de manipular *strings*, Python é uma das principais linguagens para a construção de *scripts* para o gerenciamento de dados. Como ilustramos anteriormente, existem várias APIs desenvolvidas para a linguagem Python que facilitam a coleta de dados de RSPs.

Outras ferramentas úteis na análise de dados de RSPs:

- **Estudo de grafos/redes:** Uma boa opção é o NetworkX³³. NetworkX é um pacote para Python que permite a criação, manipulação e estudo de grafos de forma bem simples;
- **Visualização de grafos:** Para essa finalidade, duas boas opções são o Gephi³⁴ e o Cytoscape³⁵. Ambos são gratuitos, e além de possibilitar a construção de boas visualizações, também oferecem suporte básico para a análise do grafo;
- **Análises estatísticas e visualização de dados em Python:** Assim como as linguagens Matlab ou R, a linguagem Python também poderia ser utilizado para realizar análises estatísticas e visualização de dados. Para que Python tenha funcionalidades

³³<https://networkx.github.io>.

³⁴<http://gephi.org>.

³⁵<http://www.cytoscape.org>.

similares ao Matlab ou R são necessários os pacotes NumPy³⁶, SciPy³⁷ e Matplotlib³⁸;

- **Processo de geocodificação (*geocoding*):** Processo de conversão de endereços (por exemplo, "1600 Amphitheatre Parkway, Mountain View, CA") em coordenadas geográficas (por exemplo, latitude 37.423021 e longitude -122.083739), que podem ser usadas para a análise espacial de dados urbanos. O Google³⁹, Bing Maps⁴⁰, Yahoo⁴¹ e Mapquest⁴² oferecem APIs para essa finalidade;
- **Visualização de dados geográficos:** Existem várias ferramentas para essa finalidade, mencionaremos aqui apenas algumas das mais populares. Google Maps⁴³, permite inserção direta de marcadores para a visualização na Web. D3⁴⁴ possibilita a visualização de dados na Web e oferece controle de baixo nível das imagens. Python Heatmaps (jjguy)⁴⁵, oferece *heatmaps* compatíveis com Google Earth baseados em Gheat. Python Heatmaps (sethoscop) ⁴⁶, oferece *heatmaps* baseados no OpenStreetMap⁴⁷.
- **Verificar existência de lei de potência (*Power law*):** Como mostramos anteriormente, ao analisar dados de RSPs podemos querer verificar se uma determinada distribuição de dados pode ser explicada por uma lei de potência. Para isso, existe uma um conjunto de ferramentas bastante útil disponibilizado em: <http://www.santafe.edu/~aaronc/powerlaws/>. Mais detalhes sobre este processo, pode ser obtido também em [Alstott et al. 2014].

2.8. Desafios

Esta seção apresenta diversos desafios sobre tópicos de pesquisa atuais relacionados com computação urbana utilizando dados de redes de sensoriamento participativo.

2.8.1. Camadas de Sensoriamento

Uma camada de sensoriamento consiste de dados descrevendo aspectos específicos de uma localização geográfica. O conceito de camada de sensoriamento é bastante amplo: ele representa dados, com seus atributos, provenientes de uma determinada fonte de dados, por exemplo uma RSP particular. Cada RSP fornece acesso aos dados relacionados a certo aspecto de uma região geográfica predefinida (por exemplo, condições de tráfego, fotos de locais, etc.), e, com isso, cada RSP distinta pode ser representada como uma camada de sensoriamento [Silva et al. 2014b].

³⁶<http://www.numpy.org>.

³⁷<http://www.scipy.org>.

³⁸<http://matplotlib.sourceforge.net>.

³⁹<https://developers.google.com/maps/documentation/geocoding/?hl=pt-br>.

⁴⁰<https://msdn.microsoft.com/en-us/library/ff701715.aspx>.

⁴¹<https://developer.yahoo.com/boss/placefinder>.

⁴²<http://www.mapquestapi.com/geocoding>.

⁴³<https://developers.google.com/maps/documentation/javascript>.

⁴⁴<http://d3js.org>.

⁴⁵<http://jjguy.com/heatmap>.

⁴⁶<http://www.sethoscop.net/heatmap>.

⁴⁷<https://www.openstreetmap.org>.

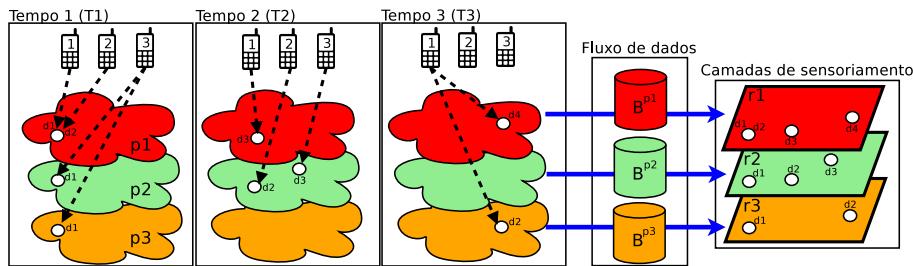


Figura 2.32. Ilustração do compartilhamento de dados em três RSPs ao longo do tempo, resultando em camadas de sensoriamento [Silva et al. 2014b].

Além de RSPs, outras fontes de dados podem ser: dados disponíveis na Web não gerados por usuários, como a condição climática fornecida pela empresa Clima Tempo⁴⁸; ou dados de redes de sensores sem fio tradicionais. Discutimos aqui o conceito de camadas de sensoriamento para as RSPs. No entanto todos os conceitos discutidos podem ser utilizados para outras fontes de dados associadas a regiões geográficas predefinidas, com as adaptações necessárias.

A Figura 2.32 ilustra o conceito de camadas de sensoriamento. Essa figura mostra dados compartilhados em três RSPs diferentes (p_1 , p_2 e p_3), por três usuários distintos em diferentes instantes de tempo. Como discutimos na Seção 2.3, esses dados devem ser coletados (por exemplo, usando uma API) e processados, passo que inclui as tarefas de análise e padronização dos dados. Cada plano na figura representa uma camada de sensoriamento para uma região específica, por exemplo o centro de Belo Horizonte, com dados provenientes de três fontes distintas. Com isso, as camadas de sensoriamento ilustradas são: *check-ins* (r_1), proveniente, por exemplo, do Foursquare; *alertas de tráfego* (r_2), proveniente, por exemplo, do Waze; e *fotos de lugares* (r_3), proveniente do Instagram, por exemplo.

O uso de camadas de sensoriamento de forma independente pode ser muito útil. Por exemplo, uma camada contendo informações de trânsito pode possibilitar a identificação em tempo real de rodovias com acidentes e buracos, cuja detecção é difícil com sensores tradicionais, mas torna-se mais factível quando os usuários participam do processo de sensoriamento. Os exemplos mencionados na Seção 2.6 também contribuem para esse ponto. No entanto, a grande motivação é realizar uma análise conjunta de múltiplas camadas para a construção de aplicações mais sofisticadas.

Sabemos que uma queixa comum dos habitantes das grandes cidades é o congestionamento. Com isso, uma aplicação que emerge naturalmente é uma que possui o objetivo de inferir as causas de congestionamento, passo fundamental para tratar o problema. Esta não é uma tarefa fácil de realizar, e o resultado pode variar entre diferentes cidades, uma vez que dependem de aspectos geográficos, culturais, econômicos, dentre outros. No entanto, a análise conjunta de diferentes camadas de sensoriamento na cidade poderia contribuir para essa aplicação. Por exemplo, poderíamos cruzar as informações fornecidas pelas camadas alertas de tráfego, *check-ins* e fotos de lugares. A primeira camada fornece dados em tempo (quase) real sobre onde estão acontecendo congestionamentos, a segunda fornece dados sobre os tipos de lugares localizados nas áreas dos congestionamentos, com isso é possível

⁴⁸<http://www.climatempo.com.br>.

entender melhor as áreas de interesse, por exemplo, identificando o tipo da área e, finalmente, através da análise da camada fotos de lugares nós podemos obter evidência visual do que acontece em tempo real próximo das áreas durante os congestionamentos. Ao analisar conjuntamente dados destas três camadas podemos detectar, por exemplo, carros bloqueando cruzamentos, e inferir as possíveis causas disso. Obviamente, outras camadas podem também ser utilizadas, tal como a condição do clima.

Há vários desafios ao lidar com dados de várias camadas simultaneamente, como os descritos a seguir.

1. **Combinação de dados:** A fim de realizar a combinação de dados nós temos que certificar que estes são consistentes em todas as camadas. Esta é uma condição obrigatória para a correta extração de informações. Por exemplo, para combinar dados compartilhados pelo mesmo usuário em diferentes camadas pode ser um problema em RSPs, pois o mesmo usuário pode participar em diferentes camadas com diferentes identificadores. Vamos supor que queremos combinar dados de um mesmo usuário compartilhados na camada *check-ins* e na camada fotos de lugares. Como os dados dessas camadas são de sistemas independentes os usuários possuem identificações diferentes. Uma forma de tentar contornar esse problema é verificar outros sistemas com o intuito de mapear o ID do usuário de uma camada em outra. Sabemos, por exemplo, que os usuários do Foursquare e Instagram tendem a ser também usuários do Twitter. Dessa forma, a chave do processo de combinação poderia ser a identificação usada no Twitter.

Além disso, os sistemas de computação urbana são, tipicamente, obrigados a responder rapidamente a consultas dos usuários (por exemplo, para prever condições de tráfego). Sem técnicas de gerenciamento de dados que permitem organizar várias camadas de sensoriamento heterogêneas, torna-se impossível realizar processos de extração de conhecimento de forma rápida. Esses são apenas alguns dos desafios relacionados a esse tópico.

2. **Validade dos dados:** Diferentes camadas podem se referir a dados válidos para diferentes intervalos de tempo. Isso é natural porque algumas fontes de dados fornecem dados em tempo (quase) real, outras não. Por exemplo, um alerta no Waze refere-se a uma situação de trânsito que pode não existir cinco minutos mais tarde. No entanto, dados do censo geralmente são válidos por um grande intervalo de tempo, meses ou anos, até o próximo censo ser publicado. Temos de estar cientes de todas essas questões ao projetar novas aplicações.

2.8.2. Dinâmica Temporal

Um aspecto pouco explorado nas análises dos dados de RSPs é o temporal. Nesse contexto, a análise de características temporais permite aprimorar tais aplicações, bem como gerar novas oportunidades de pesquisa [Gao et al. 2013, Yuan et al. 2013]. A maioria dos estudos encontrados na literatura consideram que os dados compartilhados por usuários formam RSPs estáticas, sendo a dinâmica temporal negligenciada. Isso pode acarretar em perda de informações importantes. Por exemplo, enquanto duas regiões de uma cidade podem apresentar comportamento similar nos dados agregados durante um dia, elas podem ter

diferenças quando uma perspectiva temporal é considerada na análise das atividades mais populares em cada turno.

Para ilustrar algumas iniciativas nesse sentido, em [Zhang et al. 2013] os autores analisaram atividades urbanas a partir de dados do Foursquare considerando a dinâmica temporal, a partir da divisão dos dados em períodos do dia. Os autores identificaram, por exemplo, que a atividade “comida” pode não ser tão ativa no período da tarde, mas sim, nos períodos da manhã e noite, de forma que na visão agregada não se perceba essa diferença. Essa abordagem é interessante para mostrar que determinadas atividades são pertinentes em um determinado período do dia, mas quando analisadas de forma agregada podem não ser relevantes ou podem não capturar o real comportamento dos usuários. Outro exemplo é a técnica City Image que apresentamos na Seção 2.6.1. No exemplo da Figura 2.19, a perspectiva temporal utilizada é a divisão dos dados em dias de semana/final de semana durante o dia e a noite, a partir disso foi feita uma análise dos dados por partições. Podemos perceber com o auxílio dessas imagens que existe variação significativa entre dia e noite nas duas cidades analisadas. Além disso, a imagem agregada (sem considerar partições) é bastante diferente das desagregadas, como foi mostrado em [Silva et al. 2012].

Os trabalhos descritos anteriormente fornecem indícios das vantagens da utilização de informação temporal dos dados obtidos de RSPs. No entanto, se por um lado investigar a dinâmica temporal de uma RSP é uma oportunidade para obtenção de informações mais próximas da realidade do comportamento da rede, por outro, surgem novos desafios ao adicionarmos uma dimensão temporal ao estudo, como os descritos a seguir:

1. **Janelas de tempo:** Trabalhos que analisam a questão temporal geralmente fragmentam os dados em intervalos de tempo (e.g., manhã, tarde e noite) denominados janelas. No entanto, a definição adequada do tamanho da janela é um problema, pois é necessário definir um tamanho de janela que capture dinâmicas relevantes. Nesse caso, existem inúmeras oportunidades para novas abordagens que consideram janelas com tamanhos flexíveis;
2. **Modelagem:** Geralmente, dados de uma RSP são representados como um conjunto de entidades, por exemplo, usuários ou PDIs (pontos de interesses), e suas relações (e.g., transições ou comunicação). Como a contribuição desses dados pode variar muito ao longo do tempo um modelo baseado em grafos estáticos pode não ser suficiente para capturar essa dinamicidade. Por exemplo, dados obtidos a partir do Foursquare possuem informações espaço-temporais, tais como posicionamento dos usuários e os momentos de interação. Portanto, um desafio é modelar a dinâmica espaço-temporal a fim de entender melhor, por exemplo, diversos aspectos dessa participação dos usuários. Nesse sentido, grafos temporais [Kostakos 2009] surgem como alternativa promissora que pode ser utilizada para entendimento da dinâmica espaço-temporal. Em um grafo temporal, as relações entre as entidades podem ser modeladas como arestas que podem ser criadas e destruídas ao longo do tempo. Por exemplo, entender aspectos temporais de interações entre usuários com certos locais na cidade. Dessa forma, utilizando grafos temporais para modelar uma RSP, podemos aplicar tanto conceitos de grafos (e.g., componentes conectados e caminhamento) como métricas de centralidade (e.g., centralidade de intermediação e centralidade de proximidade) para auxiliar no entendimento da dinâmica das RSPs [Nicosia et al. 2013].

2.8.3. Mecanismos de Incentivos para a Aquisição de Dados

Um ponto fundamental para as RSPs é a colaboração dos usuários, pois as aplicações em uma RSP dependem de que os usuários estejam dispostos a coletarem, processarem e transmitirem os dados sensoriados [Lee and Hoh 2010]. A colaboração entre os participantes de uma RSP reflete diretamente na qualidade e quantidade dos dados sensoriados e, consequentemente, na melhoria dos serviços oferecidos com esses dados.

No entanto, como estas aplicações consomem recursos do dispositivo do usuário, o mesmo pode ser relutante em contribuir com a rede. Diversos são os motivos que podem fazer um usuário usufruir, porém não colaborar com a RSP, tais como poupar bateria, evitar gastos com a transmissão de dados, ou mesmo por questões de privacidade [Lee and Hoh 2010].

Nos últimos anos, foram propostos dezenas de mecanismos de incentivo e realizados diversos experimentos para entender o comportamento cooperativo. Alguns desses mecanismos podem recompensar o participante por meio de pagamentos reais, virtuais ou outros prêmios. Outros mecanismos visam transformar a tarefa de sensoriamento em uma tarefa mais prazerosa e estimulante para o usuário, adicionando elementos comuns em jogos, como elementos de disputa entre os usuários (mecanismos conhecidos como baseados em *gamificação*).

Existem mecanismos que visam permitir que o usuário participe da decisão sobre a tarefa que irá realizar e sobre o pagamento que irá receber da RSP. Outros visam melhorar a qualidade dos dados obtidos e minimizar os custos com sensoriamento. Podemos mencionar também mecanismos em que o usuário negocia com a plataforma o valor da recompensa pelos dados sensoriados antes de enviá-los e, ainda, os que a plataforma decide quanto irá pagar pelos dados já enviados pelo usuário.

Um mecanismo de incentivo é eficiente se ele recruta mais participantes para a RSP e mantém esses participantes ativos no sistema. Com isso alguns dos desafios relacionados aos mecanismos de incentivos para as RSPs são:

- 1. Custos:** Para que o desenvolvimento de mecanismos de incentivo monetários sejam eficientes, deve-se considerar os custos para a plataforma da RSP e os ganhos para o participante da rede. Esses mecanismos utilizam um custo máximo para a plataforma RSP que será pago aos participantes ativos da rede. No entanto, encontrar e decidir um valor que minimize o custo para plataforma e, ao mesmo tempo, motive o usuário requer investigações futuras [Gao et al. 2015].
- 2. Validação das propostas:** A maioria das propostas de mecanismos de incentivo utilizam uma validação teórica ou pequenos experimentos controlados. No entanto, estes experimentos podem não predizer com alta precisão a participação dos usuários ao longo do tempo na plataforma. No caso de mecanismos baseados em *gamificação*, embora diversas RSPs de sucesso no mercado utilizem este conceito, aplicar uma estratégia com sucesso prévio em uma nova RSP não é garantia que funcionará. Talvez exista alguns elementos que funcionem para determinados tipos de RSPs e outros não.

2.8.4. Qualidade de Dados

A qualidade de dados é um tópico amplamente estudado pela comunidade científica. No entanto, ainda existem desafios únicos para controlar a qualidade dos dados compartilhados quando se lida com contribuições de usuários ubíquos.

Em geral, como discutimos anteriormente, os dados coletados de RSPs são, após processados, utilizados para a extração de informações contextuais, que são fundamentais para os sistemas sensíveis ao contexto [Dey and Abowd 2000].

Alguns dos principais desafios relacionados à qualidade dos dados em uma RSP são:

1. **Erros de leitura:** Um desafio que pode afetar a precisão dos dados das RSPs são possíveis erros de leitura de equipamentos. Por exemplo, um GPS pode estar mal calibrado e gerar dados cuja imprecisão está além do limite aceitável para este tipo de dado. Embora alguns erros possam parecer totalmente toleráveis, dependendo dos requisitos de uma aplicação, é possível que os limites mais restritos de precisão sejam fundamentais para sua correta operação.
2. **Ausência de estrutura:** Os dados compartilhados em RSPs, em alguns casos, são de texto livre, não apresentando uma estrutura semântica nem codificadas. Essa liberdade dada aos usuários permite que eles postem o que querem, mesmo informações incorretas, e em diferentes formatos. Por exemplo, um usuário poderia descrever um acidente em outra língua ou utilizando gírias através de algum *microblogging* como o Twitter. Com isso, o processamento dos dados se torna complexo e suscetível a erros, uma vez que há a possibilidade de dados distintos serem confundidos como um mesmo dado, ou ainda a duplicidade de dados, isto é, dados idênticos serem identificados como distintos devido a diferenças no preenchimento dos campos.
3. **Poluição dos dados:** Este desafio diz respeito à possibilidade dos dados estarem incorretos devido a um comportamento malicioso dos usuários [Coen-Porisini and Sicari 2012, Mashhadi and Capra 2011]. Podemos encontrar comportamentos maliciosos em várias esferas sociais, e o mesmo também pode ocorrer nas RSPs. Por exemplo, usuários de sistemas para compartilhamento de alertas de trânsito, como o Waze, podem gerar falsos alertas de congestionamento ou acidentes, com o intuito de incentivar os demais usuários a não utilizar determinadas vias de seu trajeto. Este comportamento malicioso poderá ocasionar em falsos positivos, por exemplo, na detecção de eventos.

2.8.5. Outros Desafios

Outra questão importante é lidar com um grande volume de dados que as RSPs podem oferecer, impondo desafios para armazenamento, processamento e indexação em tempo real usando ferramentas de gerenciamento de banco de dados tradicionais ou aplicações de processamento de dados. Isso faz com que a oferta de serviços em tempo real usando uma rede de sensoriamento participativo seja um desafio. Para resolver esta questão, precisamos de métodos para armazenar, mover e processar de forma eficaz grandes quantidades de dados. Novos paradigmas algorítmicos devem ser projetados, bem como técnicas de mineração de

dados específicas devem ser criadas de acordo com esses novos paradigmas. Outros métodos devem contemplar abordagens de engenharia de dados para grandes redes com milhões ou bilhões de nós/arestas, incluindo compressão eficaz, pesquisa e métodos para casamento de padrões [Giannotti et al. 2012].

Felizmente, a pesquisa sobre os desafios de grandes dados é muito ativa, e recentemente fez grandes avanços, por exemplo, com base em plataformas paralelas (como o Hadoop⁴⁹), para o processamento de um grande volume de dados.

Podemos ainda mencionar que as RSPs são muito dinâmicas. Usuários contam com RSPs, como o Twitter ou Waze, para transmitir seus dados sensoriados. De posse desses dados podemos extrair conhecimento. Sistemas, como o Waze, por sua vez, podem ser realimentados com esses conhecimentos obtidos e, a partir disso, eles podem fornecer informações úteis para os usuários. Esses conhecimentos também podem ser gerados por aplicativos de terceiros. Para exemplificar, na Seção 2.6 é descrito um exemplo de aplicação que permite a identificação de regiões de interesse de uma cidade. Após usar este aplicativo, os usuários podem optar por mudar o seu comportamento, como para visitar preferencialmente áreas populares, o que pode vir a afetar o número de dados compartilhados nesses locais. Isto dá uma ideia do dinamismo de uma rede de sensoriamento participativo e os desafios que surgem nessas condições.

Além desses desafios, existe ainda a questão da privacidade do usuário. Essa questão é bastante ampla e está presente em muitas camadas do sistema. Privacidade de dados em sistemas de mídia social atualmente tem sido discutida em vários estudos, como em [Pontes et al. 2012, Toch et al. 2010, Brush et al. 2010].

2.9. Oportunidades

Nesta seção ilustramos algumas das oportunidades relacionadas com o estudo de sociedades urbanas utilizando dados de RSPs.

Detecção/previsão de trânsito: De forma geral, dados de RSPs são pouco explorados em modelos de detecção/previsão de trânsito. Alguns dos trabalhos mais próximos dessa direção são: [Silva et al. 2013e, Tostes et al. 2014]. Tostes et al. analisaram condições de trânsito em relação aos dados sensoriados por duas RSPs, Foursquare e Instagram. Como vimos na Seção 2.5 os sensores de RSPs fornecem informações importantes para o melhor entendimento da dinâmica das cidades. Por exemplo, uma mensagem geolocalizada, seja no Foursquare, Instagram ou Twitter, pode ser utilizada para melhor entendermos as condições de trânsito, como foi mostrado em [Tostes et al. 2014] (trabalho discutido na Seção 2.6.5). Além disso, imagine que um usuário faça um *check-in* em casa e depois vá para o trabalho. Quando chegar no trabalho, por algum motivo, ele faz outro *check-in*. Independente se for na mesma rede social ou não, existe uma informação intrínseca no intervalo de tempo entre esses *check-ins* que consiste no desempenho do trânsito. Se o trânsito estiver mais congestionado, esse intervalo entre *check-ins* será maior do que o tempo de viagem sem congestionamento, que é facilmente calculado pela distância e velocidade máxima das vias. Além disso, os autores também levantaram várias questões nessa direção: (i) como coletar dados de mapas online em tempo real?; (ii) é possível utilizar dados de RSPs como

⁴⁹<http://hadoop.apache.org>.

uma característica para previsão de trânsito intenso?

Modelagem de camadas de sensoriamento: Temos ainda oportunidades com relação à modelagem das camadas de sensoriamento, pois numa mesma camada as entidades podem possuir relações distintas entre elas. Para ilustrar essa oportunidade considere a camada de *check-ins*. Como ilustramos anteriormente, essa camada pode representar a mobilidade urbana considerando a relação entre lugares e pessoas, sendo útil para entender, por exemplo, a frequência de transição entre diferentes lugares (entidades). Outra possibilidade é modificar a modelagem do problema, para, por exemplo, estudar as preferências de pessoas. Nesse caso, a entidade a ser analisada passa a ser o usuário. Note que dados de uma mesma camada podem ser modelados de formas distintas para responder perguntas diferentes. Alguns arcabouços, como o apresentado em [Silva et al. 2014b], oferecem suporte básico para essa questão. No entanto, existem várias oportunidades para desenvolver extensões desse arcabouço a fim de oferecer serviços mais sofisticados.

Múltiplas estratégias de mecanismos de incentivo: A maioria das propostas para incentivar a colaboração nas RSPs focam em apenas uma estratégia. Porém, como observado em [Reddy et al. 2010], a utilização de mais de uma estratégia simultaneamente pode apresentar melhores resultados. Esses autores concluem que os incentivos funcionaram melhor quando os pagamentos (recompensas) foram combinados com outros fatores, como o altruísmo do usuário e quando havia competição entre os participantes. Além disso, mostraram que um pagamento justo para todos os participantes os mantinham mais motivados do que micro pagamentos.

Confiabilidade de um determinado usuário: Dados gerados por usuários mais confiáveis provavelmente terão maior qualidade. Uma possível direção nesse sentido está relacionada com a identificação de padrões comportamentais dos usuários das RSPs. Como apresentado na Figura 2.10, quando grandes quantidades de dados são agregadas, é possível, claramente, identificar padrões de comportamento no compartilhamento de dados durante diferentes dias da semana. Assumindo que este conhecimento prévio seria uma referência do comportamento esperado dos usuários de uma dada RSP, uma possibilidade seria comparar o comportamento de um determinado usuário com este padrão de referência. Por exemplo, usuários que possuem um padrão de compartilhamento muito distinto dos demais poderia representar um usuário não confiável (e.g., um robô malicioso).

Essa abordagem discutida acima, pode ser caracterizada como uma espécie de filtragem colaborativa [Adomavicius and Tuzhilin 2005]. Esta é uma estratégia utilizada por sistemas de recomendação quando não se possui um conhecimento prévio do usuário ao qual deverá se recomendar um item. Por exemplo, utilizando as preferências de outros usuários similares a este, assumindo de que suas preferências também sejam similares.

Estudo de áreas desprivilegiadas: Regiões que fornecem uma pequena quantidade de dados em comparação com outras regiões de uma mesma cidade pode indicar uma falta de acesso à tecnologia por parte da população, uma vez que o uso de RSPs geralmente necessita *smartphones* e planos de dados 3G ou 4G, que são caros em alguns países. Os resultados preliminares sobre o uso de RSPs nesses cenários demonstram boas oportunidades para a visualização de fatos interessantes. Por exemplo, analisando os dados para a cidade do Rio de Janeiro, mostrados na Figura 2.33, observa-se que é comum encontrar áreas muito pobres ao lado de áreas ricas. Repare que a atividade de sensoriamento é pequena nas áreas indica-

das como áreas pobres. Esta informação pode ser útil de diversas formas, como para nortear melhores políticas públicas nessas áreas. A mesma informação pode ser obtida usando métodos tradicionais, tais como questionários, mas nessa possível nova maneira podemos ser capazes de obter esses dados de forma automática e mais barata usando RSPs. Para esse propósito, os algoritmos semelhantes ao proposto em [Cranshaw et al. 2012] poderiam ser aplicados.

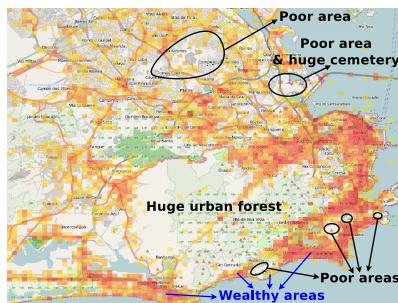


Figura 2.33. Exemplo de possível oportunidade para classificar áreas considerando a falta de dados [Silva et al. 2013a].

Classificação de áreas: Existem oportunidades de classificar áreas quando se considera conjuntamente o tempo e o local onde os *check-ins* são executadas. Pode ser possível visualizar multidões em uma cidade em tempo real. Além disso, os seres humanos possuem padrões sazonais, devido às suas rotinas. Esta sazonalidade tem um grande potencial para aplicações de previsão, uma vez que é muito provável que as pessoas repitam periodicamente suas atividades. Nós acreditamos que existem muitas oportunidades para previsão considerando o ritmo circadiano das pessoas, possibilitando a previsão, por exemplo, de multidões. Este tipo de informação é valiosa em muitos cenários, como em serviços para cidades inteligentes para evitar o tráfego em determinadas áreas e oferecer rotas alternativas para os usuários. Por exemplo, em [Hsieh et al. 2012] os autores propuseram um modelo que considera o tempo para recomendar rotas baseado em informações de RSPs para compartilhamento de localizações.

Além disso, com a utilização de dados sensoriados pelas pessoas através de RSPs é possível classificar áreas de diversas maneiras. Algumas delas foram discutidas aqui, como com relação ao cheiro, ruído e aspectos visuais. Isso pode ser útil para diversas novas aplicações e serviços. Um exemplo seria uma nova ferramenta de sugestão de rotas que sugere o menor caminho que é também o mais olfativamente agradável (as pessoas que praticam corrida de rua podem querer evitar ruas com fortes níveis de emissão de gases).

Grafos de transição urbana: Figura 2.34 mostra arestas ponderadas e nós importantes (50 maiores pesos de arestas e graus de nó) para Belo Horizonte, Cidade do México, Nova Iorque e Tóquio (considerando grafos de transição urbana, como os apresentados na seção 2.6.1, só que os nós agora são localidades individuais, ao invés de categorias de locais). Estrelas representam locais bastante visitados, setas pretas representam as arestas e círculos pretos representam *self-loops*. Quanto maior o símbolo, maior o valor. Repare que para Belo Horizonte e Cidade do México a maioria das arestas importantes são *self-loops* e de baixa distância, o que implica que as pessoas tendem a realizar atividades no bairro onde elas estão. Por outro lado, para Nova Iorque e Tóquio, cidades que são conhecidas por seus

rápidos sistemas de transporte público, existe a tendência de algumas arestas ponderadas pesadas de longa distância ao longo de ligações de transporte público. Repare o potencial para serviços que dão suporte ao planejamento urbano.

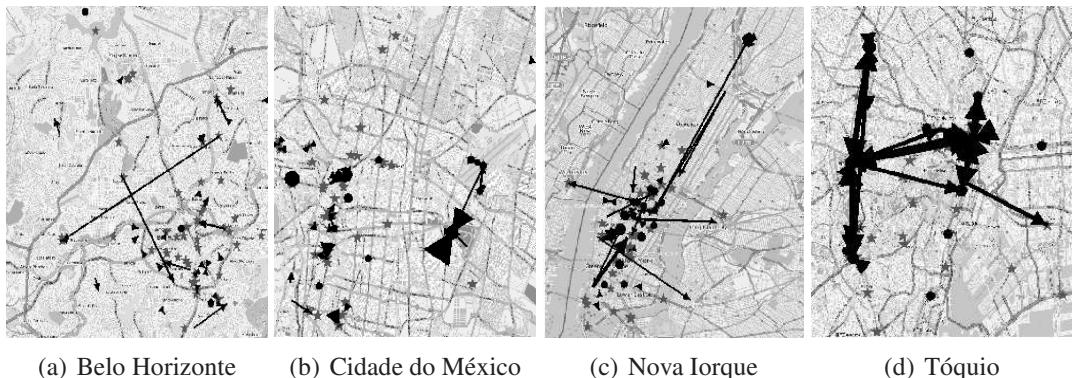


Figura 2.34. Arestas e nós importantes nos grafos de transição urbana de diversas cidades [Silva et al. 2013a].

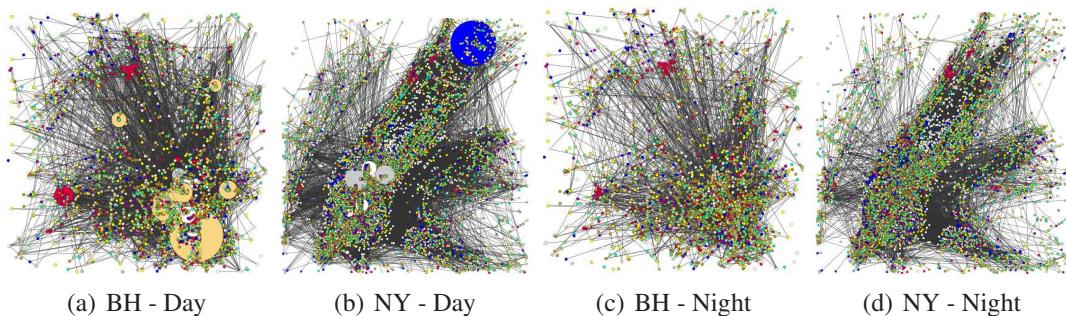


Figura 2.35. Betweenness dos nós de grafos de transição urbana para Nova Iorque e Belo Horizonte [Silva et al. 2014d].

Nessa mesma direção, podemos estudar métricas de centralidade nessa rede. Por exemplo, na Figura 2.35, mostramos os valores de centralidade de intermediação para os nós da rede. Cada cor é relacionada com uma categoria de local, e o tamanho do símbolo reflete a proporção do valor calculado. Esta abordagem pode ser utilizada para apoiar diversas aplicações, por exemplo, se for verificado um fluxo incomum e constante de pessoas entre dois locais de negócios independentes em uma cidade, os proprietários poderiam assinar um acordo comercial para aumentar suas receitas, como fazendo propaganda entre suas empresas.

Além disso, a técnica City Image, mencionada anteriormente, pode ser expandida para considerar subcategorias de locais, em vez de categorias principais. Como os dados de RSPs são altamente enviesados, algumas das transições mais populares entre as subcategorias devem ser bons indicadores da dinâmica da cidade. Essa técnica pode ser útil como uma forma de medir a distância entre duas cidades, permitindo a comparação de cidades e um agrupamento mundial que poderia ser interessante para sistemas de recomendação.

2.10. Conclusão

Neste minicurso, discutimos o conceito de computação urbana. Mostramos a relevância da área e motivamos a construção de novas aplicações para tratar questões relacionadas com a dinâmica da cidade e do comportamento social urbano. Discutimos também o sensoriamento urbano com redes de sensoriamento participativo. Mostramos que as RSPs oferecem oportunidades sem precedentes de acesso a dados de sensoriamento em escala planetária, dados que nos ajudam a entender melhor sociedades urbanas.

Estudamos também as principais técnicas utilizadas em trabalhos da área de computação urbana, bem como algumas das tecnologias e ferramentas comumente utilizadas para a análise de dados urbanos. Além disso, apresentamos diversos desafios sobre tópicos de pesquisa atuais relacionados com computação urbana utilizando dados de RSPs. Ressaltamos também várias oportunidades relacionadas ao uso de dados de RSPs em novos serviços e aplicações da área de computação urbana.

Agradecimentos

Gostaríamos de agradecer a fundamental ajuda de **Pedro O. S. Vaz de Melo e Jussara M. Almeida** para a realização deste trabalho. Gostaríamos de agradecer também aos alunos: Vinícius Mota, João Borges Neto, Clayson Celes, Felipe Cunha, Anna Ribeiro, Ana Ferreira e Virginia Kestering. Este trabalho é financiado parcialmente com o apoio das agências: CNPq, CAPES e FAPEMIG.

Referências

- [Adomavicius and Tuzhilin 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749.
- [Alstott et al. 2014] Alstott, J., Bullmore, E., and Plenz, D. (2014). powerlaw: A python package for analysis of heavy-tailed distributions. *PLoS ONE*, 9(1):e85777.
- [Balcan et al. 2009] Balcan, D., Colizza, V., Gonçalves, B., Hu, H., Ramasco, J. J., and Vespignani, A. (2009). Multiscale mobility networks and the spatial spreading of infectious diseases. *Proceedings of the National Academy of Sciences*, 106(51):21484–21489.
- [Barbosa et al. 2014] Barbosa, L., Pham, K., Silva, C., Vieira, M. R., and Freire, J. (2014). Structured open urban data: understanding the landscape. *Big data*, 2(3):144–154.
- [Barth 1969] Barth, F. (1969). *Ethnic groups and boundaries: the social organization of culture difference*. Scandinavian university books. Little, Brown.
- [Benevenuto et al. 2011] Benevenuto, F., Almeida, J. M., and Silva, A. S. (2011). Explorando redes sociais online: Da coleta e análise de grandes bases de dados às aplicações. *Proc. of SBRC'11*, pages 63–94.
- [Blei et al. 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.
- [Bollen et al. 2011] Bollen, J., Mao, H., and Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8.
- [Brenner and Smith 2013] Brenner, J. and Smith, A. (2013). 72% of online adults are social networking site users. <http://goo.gl/HTgNy3>.
- [Brockmann et al. 2006] Brockmann, D., Hufnagel, L., and Geisel, T. (2006). The scaling laws of human travel. *Nature*, 439(7075):462–465.

- [Brush et al. 2010] Brush, A. B., Krumm, J., and Scott, J. (2010). Exploring end user preferences for location obfuscation, location-based services, and the value of location. In *Proc. of Ubicomp '10*, pages 95–104, Copenhagen, Denmark. ACM.
- [Burke et al. 2006] Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. B. (2006). Participatory sensing. In *Proc. of Workshop on World-Sensor-Web (WSW'06)*, pages 117–134, Boulder, USA.
- [Burt 1992] Burt, R. S. (1992). *Structural Holes: The Social Structure of Competition*. Harvard University Press.
- [CENS/UCLA] CENS/UCLA. *Participatory Sensing / Urban Sensing Projects*. <http://research.cens.ucla.edu/>.
- [Cha et al. 2010] Cha, M., Haddadi, H., Benevenuto, F., and Gummadi, K. (2010). Measuring user influence in twitter: The million follower fallacy. In *Proc. of ICWSM'10*, Washington, USA.
- [Chatterjee and Hadi 2015] Chatterjee, S. and Hadi, A. S. (2015). *Regression analysis by example*. John Wiley & Sons.
- [Cheng et al. 2011] Cheng, Z., Caverlee, J., Lee, K., and Sui, D. Z. (2011). Exploring Millions of Footprints in Location Sharing Services. In *Proc. of ICWSM'11*, Barcelona, Spain.
- [Cho et al. 2011] Cho, E., Myers, S. A., and Leskovec, J. (2011). Friendship and mobility: user movement in location-based social networks. In *Proc. KDD '11*, pages 1082–1090, San Diego, USA. ACM.
- [Clauset et al. 2009] Clauset, A., Shalizi, C. R., and Newman, M. E. J. (2009). Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703.
- [Coen-Porisini and Sicari 2012] Coen-Porisini, A. and Sicari, S. (2012). Improving data quality using a cross layer protocol in wireless sensor networks. *Comput. Netw.*, 56(17):3655–3665.
- [Crandall et al. 2009] Crandall, D. J., Backstrom, L., Huttenlocher, D., and Kleinberg, J. (2009). Mapping the world's photos. In *Proc. of WWW '09*, pages 761–770, Madrid, Spain. ACM.
- [Cranshaw et al. 2012] Cranshaw, J., Schwartz, R., Hong, J. I., and Sadeh, N. (2012). The Livehoods Project: Utilizing Social Media to Understand the Dynamics of a City. In *Proc. of ICWSM'12*, Dublin, Ireland.
- [Dey and Abowd 2000] Dey, A. K. and Abowd, G. D. (2000). Towards a Better Understanding of Context and Context-Awareness. In *Proc. of CHI 2000 Workshops*, The Hague, The Netherlands.
- [D'Hondt et al. 2013] D'Hondt, E., Stevens, M., and Jacobs, A. (2013). Participatory noise mapping works! an evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring. *Pervasive and Mobile Computing*, 9(5):681–694.
- [Duggan and Smith 2014] Duggan, M. and Smith, A. (2014). Social media update 2013. <http://goo.gl/JhuiOG>.
- [Easley and Kleinberg 2010] Easley, D. and Kleinberg, J. (2010). *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press.
- [Eric Paulos and Townsend 2004] Eric Paulos, K. A. and Townsend, A. (2004). Ubicomp in the urban frontier. In *Workshop at Ubicomp'04*, Nottingham, UK.
- [Ester et al. 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD-96*, Portland, USA.
- [Freedman 2009] Freedman, D. A. (2009). *Statistical models: theory and practice*. cambridge university press.
- [Ganti et al. 2011] Ganti, R., Ye, F., and Lei, H. (2011). Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39.
- [Gao et al. 2015] Gao, H., Liu, C., Wang, W., Zhao, J., Song, Z., Su, X., Crowcroft, J., and Leung, K. (2015). A survey of incentive mechanisms for participatory sensing. *Communications Surveys Tutorials, IEEE*, PP(99):1–1.

- [Gao et al. 2013] Gao, H., Tang, J., Hu, X., and Liu, H. (2013). Exploring temporal effects for location recommendation on location-based social networks. In *Proc. of RecSys '13*, pages 93–100, Hong Kong, China.
- [Giannotti et al. 2012] Giannotti, F., Pedreschi, D., Pentland, A., Lukowicz, P., Kossman, D., Crowley, J., and Helbing, D. (2012). A planetary nervous system for social mining and collective awareness. *The Eur. Phy. Jour. Special Topics*, 214(1):49–75.
- [Gomide et al. 2011] Gomide, J., Veloso, A., Jr., W. M., Almeida, V., Benevenuto, F., Ferraz, F., and Teixeira, M. (2011). Dengue surveillance based on a computational model of spatio-temporal locality of twitter. In *Proc. of WebSci'11*, Evanston, USA.
- [Gonzalez et al. 2008] Gonzalez, M. C., Hidalgo, C. A., and Barabasi, A.-L. (2008). Understanding individual human mobility patterns. *Nature*, 453(7196):779–782.
- [Gonçalves et al. 2013] Gonçalves, P., Araújo, M., Benevenuto, F., and Cha, M. (2013). Comparing and combining sentiment analysis methods. In *Proceedings of the 1st ACM Conference on Online Social Networks (COSN'13)*, Boston, USA.
- [Harrell 2013] Harrell, F. E. (2013). *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*. Springer Science & Business Media.
- [Hartigan and Wong 1979] Hartigan, J. A. and Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108.
- [Hochman and Schwartz 2012] Hochman, N. and Schwartz, R. (2012). Visualizing instagram: Tracing cultural visual rhythms. In *Proc. of ICWSM'12*, pages 6–9, Dublin, Ireland. AAAI.
- [Hsieh et al. 2012] Hsieh, H.-P., Li, C.-T., and Lin, S.-D. (2012). Exploiting large-scale check-in data to recommend time-sensitive routes. In *Proc. of UrbComp '12*, pages 55–62, Beijing, China. ACM.
- [Instagram 2014] Instagram (2014). Instagram today: 200 million strong. <http://blog.instagram.com/post/80721172292/200m>.
- [Jolliffe 2002] Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer, second edition.
- [Joseph et al. 2012] Joseph, K., Tan, C. H., and Carley, K. M. (2012). Beyond local, categories and friends: clustering foursquare users with latent topics. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 919–926, Pittsburgh, USA. ACM.
- [Karamshuk et al. 2013] Karamshuk, D., Noulas, A., Scellato, S., Nicosia, V., and Mascolo, C. (2013). Geo-spotting: Mining online location-based services for optimal retail store placement. In *Proc. of KDD '13*, pages 793–801, Chicago, Illinois, USA. ACM.
- [Kaufman and Rousseeuw 2009] Kaufman, L. and Rousseeuw, P. J. (2009). *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons.
- [Kindberg et al. 2007] Kindberg, T., Chalmers, M., and Paulos, E. (2007). Guest editors' introduction: Urban computing. *IEEE Pervasive Computing*, 6(3):18–20.
- [Kisilevich et al. 2010] Kisilevich, S., Krstajic, M., Keim, D., Andrienko, N., and Andrienko, G. (2010). Event-based analysis of people's activities and behavior using flickr and panoramio geotagged photo collections. In *Proc. of Conf. on Inf. Vis.*, pages 289–296, London, UK. IEEE.
- [Kostakos 2009] Kostakos, V. (2009). Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023.
- [Kostakos and O'Neill 2008] Kostakos, V. and O'Neill, E. (2008). {City ware: Urban Computing to Bridge Online}. *Handbook of Research on Urban Informatics: The Practice and Promise of the Real-Time City*.
- [Krings et al. 2009] Krings, G., Calabrese, F., Ratti, C., and Blondel, V. D. (2009). Urban gravity: a model for inter-city telecommunication flows. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(07):L07003.
- [Lane et al. 2010] Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A. (2010). A survey of mobile phone sensing. *Comm. Mag., IEEE*, 48(9):140–150.

- [Lane et al. 2008] Lane, N. D., Eisenman, S. B., Musolesi, M., Miluzzo, E., and Campbell, A. T. (2008). Urban sensing systems: Opportunistic or participatory? In *Proc. of HotMobile '08*, pages 11–16, Napa Valley, California. ACM.
- [Lee and Hoh 2010] Lee, J.-S. and Hoh, B. (2010). Dynamic pricing incentive for participatory sensing. *Pervasive and Mobile Computing*, 6(6):693–708.
- [Loureiro et al. 2003] Loureiro, A. A. F., Nogueira, J. M. S., Ruiz, L. B., Mini, R. A., Nakamura, E. F., and Figueiredo, C. M. S. (2003). Redes de sensores sem fio. *Proc. of SBRC'03*, pages 179–226.
- [Luxburg 2007] Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.
- [MacKerron and Mourato 2013] MacKerron, G. and Mourato, S. (2013). Happiness is greater in natural environments. *Global Environmental Change*, 23(5):992–1000.
- [Maisonneuve et al. 2009] Maisonneuve, N., Stevens, M., Niessen, M. E., and Steels, L. (2009). Noisetube: Measuring and mapping noise pollution with mobile phones. In *Information Technologies in Environmental Engineering*, pages 215–228. Springer.
- [Martine et al. 2007] Martine, G., Marshall, A., et al. (2007). State of world population 2007: unleashing the potential of urban growth. In *State of world population 2007: unleashing the potential of urban growth*. UNFPA.
- [Mashhadi and Capra 2011] Mashhadi, A. J. and Capra, L. (2011). Quality Control for Real-time Ubiquitous Crowdsourcing. In *Proc. of UbiCrowd'11*, pages 5–8, Beijing, China.
- [Nazir et al. 2008] Nazir, A., Raza, S., and Chuah, C.-N. (2008). Unveiling facebook: A measurement study of social network based applications. In *Proc. of IMC '08*, pages 43–56, Vouliagmeni, Greece.
- [Newman 2010] Newman, M. (2010). *Networks: an introduction*. Oxford University Press, Inc.
- [Newman 2003] Newman, M. E. J. (2003). The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256.
- [Ng et al. 2002] Ng, A. Y., Jordan, M. I., Weiss, Y., et al. (2002). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856.
- [Nguyen and Szymanski 2012] Nguyen, T. and Szymanski, B. K. (2012). Using location-based social networks to validate human mobility and relationships models. *arXiv preprint arXiv:1208.3653*.
- [Nicosia et al. 2013] Nicosia, V., Tang, J., Mascolo, C., Musolesi, M., Russo, G., and Latora, V. (2013). Graph metrics for temporal networks. In *Temporal Networks*, pages 15–40. Springer.
- [Noulas et al. 2011a] Noulas, A., Scellato, S., Mascolo, C., and Pontil, M. (2011a). An Empirical Study of Geographic User Activity Patterns in Foursquare. In *Proc. of ICWSM'11*, Barcelona, Spain.
- [Noulas et al. 2011b] Noulas, A., Scellato, S., Mascolo, C., and Pontil, M. (2011b). Exploiting Semantic Annotations for Clustering Geographic Areas and Users in Location-based Social Networks. In *Proc. of the Fifth Int'l Conf. on Weblogs and Social Media (ICWSM'11)*, Barcelona, Spain. AAAI.
- [Paulos and Goodman 2004] Paulos, E. and Goodman, E. (2004). The familiar stranger: anxiety, comfort, and play in public places. In *Proc. of CHI'04*, pages 223–230, Vienna, Austria. ACM.
- [Poblete et al. 2011] Poblete, B., Garcia, R., Mendoza, M., and Jaimes, A. (2011). Do all birds tweet the same?: characterizing twitter around the world. In *Proc. of CIKM*, pages 1025–1030, Glasgow, UK. ACM.
- [Pontes et al. 2012] Pontes, T., Magno, G., Vasconcelos, M., Gupta, A., Almeida, J., Kumaraguru, P., and Almeida, V. (2012). Beware of what you share: Inferring home location in social networks. In *Proc. of ICDMW*, pages 571–578, Brussels, Belgium.
- [Quercia et al. 2012] Quercia, D., Capra, L., and Crowcroft, J. (2012). The social world of twitter: Topics, geography, and emotions. In *Proc. of ICWSM'12*, Dublin, Ireland.

- [Quercia et al. 2014] Quercia, D., Schifanella, R., and Aiello, L. M. (2014). The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*, HT '14, pages 116–125, New York, NY, USA. ACM.
- [Quercia et al. 2015] Quercia, D., Schifanella, R., Aiello, L. M., and McLean, K. (2015). Smelly maps: The digital life of urban smellscapes. In *Proc. of ICWSM'15*, Oxford, UK.
- [Reddy et al. 2010] Reddy, S., Estrin, D., Hansen, M., and Srivastava, M. (2010). Examining micro-payments for participatory sensing data collections. In *Proc. of Ubicomp '10*, pages 33–36, Copenhagen, Denmark. ACM.
- [Reis et al. 2015] Reis, J., Benevenuto, F., Vaz de Melo, P., Prates, R., Kwak, H., and An, J. (2015). Breaking the news: First impressions matter on online news. In *Proceedings of the 9th International AAAI Conference on Web-Blogs and Social Media*, Oxford, UK.
- [Sakaki et al. 2010a] Sakaki, T., Okazaki, M., and Matsuo, Y. (2010a). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proc. of WWW'10*, pages 851–860, Raleigh, USA.
- [Sakaki et al. 2010b] Sakaki, T., Okazaki, M., and Matsuo, Y. (2010b). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proc. of WWW'10*, pages 851–860, Raleigh, USA. IW3C2.
- [Shannon 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell system tech. jour.*, 27.
- [Silva et al. 2014a] Silva, T., Vaz De Melo, P., Almeida, J., and Loureiro, A. (2014a). Large-scale study of city dynamics and urban social behavior using participatory sensing. *Wireless Communications, IEEE*, 21(1):42–51.
- [Silva et al. 2014b] Silva, T. H., Vaz de Melo, P., Almeida, J., Viana, A., Salles, J., and Loureiro, A. (2014b). Participatory Sensor Networks as Sensing Layers. In *Proc. of SocialCom'14*, Sydney, Australia.
- [Silva et al. 2012] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., and Loureiro, A. A. F. (2012). Visualizing the invisible image of cities. In *Proc. IEEE CPScom'12*, pages 382–389, Besancon, France.
- [Silva et al. 2013a] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., and Loureiro, A. A. F. (2013a). Challenges and opportunities on the large scale study of city dynamics using participatory sensing. In *Proc. of IEEE ISCC'13*, pages 528–534, Split, Croatia.
- [Silva et al. 2013b] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., and Loureiro, A. A. F. (2013b). Uma Fotografia do Instagram: Caracterização e Aplicação. In *Proc. of SBRC'13*, Brasília, Brazil.
- [Silva et al. 2014c] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., Musolesi, M., and Loureiro, A. A. F. (2014c). You are What you Eat (and Drink): Identifying Cultural Boundaries by Analyzing Food & Drink Habits in Foursquare. In *Proc. of ICWSM'14*, Ann Arbor, USA.
- [Silva et al. 2013c] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., Salles, J., and Loureiro, A. A. F. (2013c). A picture of Instagram is worth more than a thousand words: Workload characterization and application. In *Proc. of DCROSS'13*, pages 123–132, Cambridge, USA.
- [Silva et al. 2013d] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., Salles, J., and Loureiro, A. A. F. (2013d). A comparison of foursquare and instagram to the study of city dynamics and urban social behavior. In *Proc. of UrbComp'13*, pages 1–8, Chicago, USA.
- [Silva et al. 2014d] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., Salles, J., and Loureiro, A. A. F. (2014d). Revealing the city that we cannot see. *ACM Trans. Internet Technol.*, 14(4):26:1–26:23.
- [Silva et al. 2013e] Silva, T. H., Vaz de Melo, P. O. S., Viana, A., Almeida, J. M., Salles, J., and Loureiro, A. A. F. (2013e). Traffic Condition is more than Colored Lines on a Map: Characterization of Waze Alerts. In *Proc. of SocInfo'13*, pages 309–318, Kyoto, Japan.
- [Sinnott 1984] Sinnott, R. W. (1984). Virtues of the Haversine. *Sky and Telescope*, 68(2):159+.
- [Sørensen 1948] Sørensen, T. (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biologiske Skrifter*, 5(4).

- [Srivastava et al. 2012] Srivastava, M., Abdelzaher, T., and Szymanski, B. (2012). Human-centric sensing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):176–197.
- [Thelwall et al. 2010] Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. (2010). Sentiment in short strength detection informal text. *J. Am. Soc. Inf. Sci. Technol.*, 61(12):2544–2558.
- [Toch et al. 2010] Toch, E., Cranshaw, J., Drielsma, P. H., Tsai, J. Y., Kelley, P. G., Springfield, J., Cranor, L., Hong, J., and Sadeh, N. (2010). Empirical models of privacy in location sharing. In *Proc. of Ubicomp’10*, pages 129–138, Copenhagen, Denmark. ACM.
- [Tostes et al. 2013] Tostes, A. I. J., Duarte-Figueiredo, F., Assunção, R., Salles, J., and Loureiro, A. A. F. (2013). From data to knowledge: City-wide traffic flows analysis and prediction using bing maps. In *Proc. of ACM UrbComp’13*, Chicago, USA.
- [Tostes et al. 2014] Tostes, A. I. J., Silva, T. H., Duarte-Figueiredo, F., and Loureiro, A. A. F. (2014). Studying traffic conditions by analyzing foursquare and instagram data. In *Proc. of ACM PE-WASUN’14*, Montreal, Canada.
- [Yan 2009] Yan, X. (2009). *Linear regression analysis: theory and computing*. World Scientific.
- [Youyou et al. 2015] Youyou, W., Kosinski, M., and Stillwell, D. (2015). Computer-based personality judgments are more accurate than those made by humans. *Nat. Acad. of Sci.*, 112(4):1036–1040.
- [Yuan et al. 2013] Yuan, Q., Cong, G., Ma, Z., Sun, A., and Thalmann, N. M. (2013). Time-aware point-of-interest recommendation. In *Proc. of SIGIR ’13*, pages 363–372, Dublin, Ireland. ACM.
- [Zaki and Meira Jr 2014] Zaki, M. J. and Meira Jr, W. (2014). *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press.
- [Zhang et al. 2013] Zhang, K., Jin, Q., Pelechrinis, K., and Lappas, T. (2013). On the importance of temporal dynamics in modeling urban activity. In *Proc. of UrbComp’13*, pages 7:1–7:8, Chicago, Illinois.
- [Zheng et al. 2014a] Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014a). Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):38.
- [Zheng et al. 2014b] Zheng, Y., Liu, T., Wang, Y., Zhu, Y., Liu, Y., and Chang, E. (2014b). Diagnosing new york city’s noises with ubiquitous data. In *Proc. of UbiComp’14*, pages 715–725, Seattle, Washington. ACM.
- [Zheng et al. 2009] Zheng, Y., Zhang, L., Xie, X., and Ma, W.-Y. (2009). Mining interesting locations and travel sequences from gps trajectories. In *Proc. of WWW’09*, pages 791–800, Madrid, Spain. ACM.
- [Zheng et al. 2012] Zheng, Y.-T., Zha, Z.-J., and Chua, T.-S. (2012). Mining travel patterns from geotagged photos. *ACM Trans. Intell. Syst. Technol.*, 3(3):56:1–56:18.
- [Zipf 1946] Zipf, G. K. (1946). The p1 p2/d hypothesis: On the intercity movement of persons. *American sociological review*, pages 677–686.

Chapter

3

Introdução à Otimização Combinatória

Flávio K. Miyazawa e Cid C. de Souza

Abstract

Most of the computational problems are discrete in nature and involve the search of a solution satisfying certain properties, with possible alternatives growing in a combinatorial way. A classic example is the sorting problem, where one must find a permutation of a sequence in non-decreasing order. Moreover, real problems of this type become more complex, when we consider that available resources are limited and the need to optimize their use.

Consider, for example, a city road network and a set of bus stops where employees of a company pick up a bus having sufficient capacity to transport them all. The bus route starts in the garage, pass through the points and take the employees to the work place. However, when we consider the transportation cost, usually proportional to the total traveled distance, it is desirable that the bus go through the shortest route. This is a typical combinatorial optimization problem. In addition to vehicle routing problems, several other applications can be found in various fields of knowledge.

In this mini-course, we present some of the main algorithmic techniques for combinatorial optimization problems.

Resumo

Grande parte dos problemas computacionais são de natureza discreta e envolvem a busca por uma solução atendendo certas propriedades, sendo que as possíveis alternativas crescem de forma combinatória. Um exemplo clássico é o problema de ordenação, onde se deseja encontrar uma permutação de uma sequência com ordenação não-decrescente. Além disso, problemas reais deste tipo tornam-se ainda mais complexos quando existe limitação nos recursos disponíveis e se quer otimizar alguma medida específica.

Considere, por exemplo, a malha viária de uma cidade e pontos onde funcionários tomam um ônibus fretado com capacidade suficiente para transportá-los até uma empresa. O ônibus deve sair da garagem, pegar os funcionários nos pontos, e levá-los ao trabalho. Sendo a malha conexa, qualquer permutação dos pontos define uma possível rota para o ônibus. Porém, quando consideramos o custo do transporte, usualmente proporcional à distância total percorrida pelo veículo, é desejável que o ônibus siga por uma rota de menor percurso. Este é um típico problema de otimização combinatória. Além do roteamento de veículos, inúmeras outras aplicações podem ser encontradas nas mais diversas áreas do conhecimento.

O texto apresenta algumas das principais técnicas algorítmicas para tratamento de problemas de otimização combinatória.

3.1. Introdução

O texto contempla uma breve introdução à área de Otimização Combinatória. Trata-se de uma área que engloba grande quantidade de problemas e que busca por soluções que façam melhor uso dos recursos envolvidos. Nestes problemas, não basta ter uma das soluções possíveis, mas uma que também otimize os objetivos de interesse. Alguns objetivos típicos dos problemas de otimização combinatória, consistem no geral em aproveitar melhor os materiais no processo de produção, otimizar o tempo para realização de ações e operações, transportar mais materiais pelas melhores rotas, diminuir chances de se obter prejuízos, aumentar lucros e diminuir gastos, etc. Para ilustrar a diversidade de áreas, os problemas ocorrem no projeto de redes de telecomunicações, projeto de redes de escoamento de produtos, projeto de circuitos VLSI, corte de materiais, empacotamento em containers, localização de centros distribuidores, escalonamento de tarefas, roteamento de veículos, sequenciamento de DNA, etc.

Os problemas de otimização combinatória podem ser de minimização ou de maximização. Em ambos os casos, temos uma função aplicada a um domínio finito, que em geral é enumerável. Apesar de finito, o domínio da função é em geral grande e algoritmos ingênuos que verificam cada elemento deste domínio se tornam impraticáveis. Desta maneira, surge a necessidade de usar técnicas mais elaboradas para encontrar soluções de valor ótimo, que pode ser de valor mínimo, caso o problema seja de minimização, ou máximo, caso o problema seja de maximização.

Neste texto apresentamos uma breve introdução a algumas técnicas e conceitos básicos da área de Otimização Combinatória.

Alguns problemas de otimização combinatória podem ser bem resolvidos, mas para muitos outros não são esperados algoritmos exatos e rápidos para problemas de grande porte. Muitos destes problemas são importantes e necessitam de soluções, mesmo que não sejam de valor ótimo (de preferência que tenham valor próximo do ótimo) ou sejam obtidas após muito processamento computacional.

O texto está organizado da seguinte forma. Na Seção 3.1.1 a seguir, apresenta-se

a notação usada no texto. Na Seção 3.3 apresenta-se a técnica de projetar algoritmos usando indução matemática. Nas seções 3.4 e 3.5 discutem-se, respectivamente os algoritmos gulosos e os de programação dinâmica, enfatizando sua aplicações na resolução de problemas de Otimização Combinatória. A Seção 3.6 é dedicada a uma discussão sobre Programação Linear e Programação Inteira. As seções 3.7 e 3.8 tratam do desenvolvimento de algoritmos enumerativos usando, respectivamente, os paradigmas de *backtracking* e de *branch and bound*. Finalmente, na Seção 3.9 são feitas algumas considerações finais sobre técnicas e abordagens que não puderam ser contempladas no texto, mas seria interessante contemplá-las em um primeiro curso.

Por ser um texto resumido, é recomendado que o leitor consulte outros livros na literatura, como [Cormen et al. 2009, Dasgupta et al. 2008, Ferreira e Wakabayashi 1996, Manber 1989, Nemhauser e Wolsey 1988, Papadimitriou e Steiglitz 1982, Schrijver 1986, Szwarcfiter 1988, Wolsey 1998] e [Ziviani 2011].

3.1.1. Notação

Conjuntos, Vetores e Funções

Denota-se por \mathbb{N} (resp. \mathbb{Q} e \mathbb{R}) o conjunto dos números naturais $\{1, 2, \dots\}$ (resp. conjunto dos números racionais e reais).

Denota-se por $\mathbf{0}^d$ o vetor d -dimensional contendo apenas zeros. Quando a dimensão estiver clara pelo contexto, tal vetor será denotado apenas por $\mathbf{0}$. Dados vetores $u = (u_1, \dots, u_n) \in \mathbb{R}^n$ e $v = (v_1, \dots, v_m) \in \mathbb{R}^m$, para inteiros positivos n e m , o *produto interno* $u \cdot v$ entre u e v será denotado por $u \cdot v = \sum_{i=1}^{\min\{n,m\}} u_i v_i$ e a *concatenação* de u e v é dada pelo vetor $u \| v = (u_1, \dots, u_n, v_1, \dots, v_m)$.

Dada uma função numérica $f : D \rightarrow \mathbb{R}$, denota-se por $f(S)$ a soma dos valores de f nos elementos de D , i.e., $f(S) := \sum_{e \in S} f(e)$ para todo $S \subseteq D$. Dado conjuntos S e T , denota-se por $S - T$ (resp. $S + T$) o conjunto $S \setminus T$ (resp. $S \cup T$). O conjunto formado por apenas um elemento $\{e\}$ pode ser representado apenas pelo elemento e . Assim, $S - e$ representa o conjunto $S \setminus \{e\}$.

Complexidade Assintótica

Ao medir a complexidade de tempo dos algoritmos, a maneira padrão é defini-la em termos do tamanho (*e.g.* número de bits) da entrada. No caso, deve-se apresentar uma função que, dado o tamanho da entrada, devolve o número de passos do algoritmo dentro do modelo computacional considerado. Na maioria das vezes, basta saber a ordem de grandeza destes números, sem no entanto saber exatamente o número de passos ou ciclos realizados para cada entrada. Ademais, a codificação de um algoritmo em linguagem de máquina para dois computadores diferentes, pode levar a quantidade diferente de ciclos.

Assim, na análise da complexidade de tempo de um algoritmo, será usada a no-

tação assintótica, que permite restringir a análise nos termos que crescem, de maneira assintótica, mais rapidamente, sem se preocupar com constantes multiplicativas ou termos menores. Por exemplo, se o número de ciclos em um computador for dado por uma função $T(n) = 10n^2 + 1000n + 20$, o termo que cresce mais rápido e domina os outros termos é $10n^2$. Por mais que termos de ordem menor tenham valores grandes no início o termo de ordem maior, como o $10n^2$ domina os demais a medida que n cresce. Para isto, note que $\frac{T(n)}{10n^2}$ tende a 1 a medida que n cresce. Desconsiderando as constantes multiplicativas, que podem ser diferentes de um computador para o outro, temos que $T(n)$ é da ordem de n^2 , ou como definido abaixo, $T(n) \in O(n^2)$.

Dada função não negativa $g(n)$, denotamos por $O(g(n))$ o conjunto $O(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tal que } 0 \leq f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$, por $\Omega(g(n))$ o conjunto $\Omega(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tal que } 0 \leq cg(n) \leq f(n) \text{ para todo } n \geq n_0\}$ e por $\Theta(g(n))$ o conjunto $\Theta(g(n)) = \{f(n) : \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tal que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ para todo } n \geq n_0\}$.

Para funções que tem crescimento muito rápido, como as funções exponenciais em n , é comum utilizar a notação $O^*(g(n))$, definida como o conjunto $O^*(g(n)) = \{f(n) : \text{existe constante } c \text{ tal que } f(n) \in O(g(n)n^c)\}$. Esta notação remove os polinômios multiplicativos. Para entender porque a notação se justifica, considere uma função $f(n) = p(n)a^n$, onde $p(n)$ é um polinômio e a uma constante maior que 1. Note que $f(n)$ é assintoticamente menor que a função $g(n) = (a + \varepsilon)^n$ para qualquer polinômio $p(n)$ e constante $\varepsilon > 0$ (que pode ser bem pequena).

Teoria dos Grafos

Um *grafo* $G = (V, E)$ é uma estrutura composta por dois tipos de objetos, os *vértices*, dado por um conjunto V , e as *arestas*, dado por um conjunto E tal que $E \subseteq \{\{u, v\} : u, v \in V\}$ e $V \cap E = \emptyset$.

Se $G = (V, E)$ e $e = \{u, v\} \in E$ é uma aresta de G , então, u e v são *adjacentes* entre si e ditas *extremidades* de e . Neste caso, e é dito incidir em u e em v . Note que nesta definição não é permitido ter mais do que uma aresta contendo dois vértices adjacentes ou ter uma aresta incidente em apenas um vértice (estes grafos também são conhecidos como *grafos simples*). O conjunto dos vértices adjacentes a v é dado por $\text{Adj}(v)$. Dado vértice $v \in V$ (resp. conjunto $S \subseteq V$), o conjunto das arestas com exatamente uma extremidade em v (resp. em S) é denotado por $\delta(v)$ (resp. $\delta(S)$). O *grau* de v é o número de arestas que incidem em v . Um grafo pode ser representado de maneira gráfica em um plano, fazendo com que cada vértice seja um ponto no plano e vértices adjacentes são ligados por uma linha. Algumas operações usadas para conjuntos são estendidas para grafos. Se $G = (V, E)$ é um grafo, então $v \in G$ (resp. $e \in G$) denota o mesmo que $v \in V$ (resp. $e \in E$). O grafo $G - v$ é o grafo obtido a partir de G removendo-se v e as arestas incidentes a ele. Da mesma forma, para um conjunto de vértices $S \subseteq V$, o grafo $G - S$

é o grafo obtido a partir de G removendo cada vértice de S . Se $\mathcal{S} = \{S_1, \dots, S_k\}$ são subconjuntos disjuntos de V então $E(S_1, \dots, S_k)$ é o conjunto de arestas com extremos em partes distintas. Em particular, se \mathcal{S} é uma partição de V com k partes, o conjunto \mathcal{S} é dito ser um k -corte de G . Um 2-corte também é chamado pelo termo *corte*.

Dado grafo $G = (V, E)$, o grafo $H = (V_H, E_H)$ é um *subgrafo* de G se $V_H \subseteq V$ e $E_H \subseteq E$. Um *passeio* em um grafo $G = (V, E)$ é uma sequência $P = (v_0, e_1, v_1, \dots, e_t, v_t)$, onde $v_i \in V$, $e_i \in E$ e $e_i = \{v_{i-1}, v_i\}$ para $i = 1, \dots, t$. Neste caso, P é um passeio de v_0 a v_t , sendo estes também seus *extremos*. Um *caminho* em G é um passeio onde todos os vértices são distintos. Um *ciclo* é um passeio onde todos os vértices são distintos, exceto o último que é igual ao primeiro. Quando o interesse for apenas no conjunto de vértices e arestas destes grafos, estas estruturas poderão ser consideradas simplesmente como grafos ou subgrafos, pelo conjunto de vértices e arestas. Um grafo é dito *conexo* se para todo par de vértices u, v em V , existe um caminho em G de u a v , caso contrário, é dito *desconexo*. Uma *árvore* é um grafo conexo sem ciclos. Um subgrafo que é ciclo ou caminho de G é dito *hamiltoniano* se contém todos os vértices de G . Um subgrafo H de G é dito *gerador* se H contém todos os vértices de G .

Para muitos problemas apresentados, os grafos tem pesos ou custos nas arestas. Assim, a tupla $G = (V, E, c)$ representa um grafo ponderado nas arestas, onde $c(e)$ (ou c_e) é o peso ou custo da aresta e .

Um *grafo orientado* (ou *direcionado*) $G = (V, A)$ é uma estrutura dada por dois tipos de objetos, os vértices, dado pelo conjunto V , e os arcos, dado pelo conjunto A , tal que $A \subseteq V \times V$ e $V \cap A = \emptyset$. Diferente dos grafos (não-orientados), um arco $a = (u, v)$, é um par ordenado. Neste caso, o vértice u é a *origem* de a e v é o destino de a , ou analogamente, diz-se que a *sai* de u e *entra* em v . Denote por $\text{Adj}^+(u)$ o conjunto de vértices v para os quais há um arco $(u, v) \in A$. Dado conjunto de vértices $S \subseteq V$, denota-se $\delta^+(S)$ (resp. $\delta^-(S)$) o conjunto de arcos que saem de (resp. entram em) algum vértice de S e entram em (resp. saem de) algum vértice de $V \setminus S$.

Dado um grafo orientado $G = (V, A)$, o grafo $H = (V_H, A_H)$ é um *subgrafo* de G se $V_H \subseteq V$ e $A_H \subseteq A$. As definições de passeios, caminhos e ciclos, são análogas ao caso não-orientado, porém respeitando uma orientação das arestas. Um *passeio* em um grafo orientado $G = (V, A)$ é uma sequência $P = (v_0, a_1, v_1, \dots, a_t, v_t)$, onde $v_i \in V$, $a_i \in A$ e $a_i = (v_{i-1}, v_i)$ para $i = 1, \dots, t$. Neste caso, P é um passeio de v_0 a v_t , sendo estes também seus *extremos*. As definições de caminhos e ciclos, bem como as versões hamiltonianas, para grafos orientados são definidas analogamente.

Ao leitor interessado em saber mais sobre a teoria dos grafos, recomenda-se os livros [Feofiloff et al. 2004] e [Bondy e Murty 2008].

3.2. Otimização Combinatória

Considere o seguinte problema. Suponha que uma empresa, chamada BUBBLE, é dona de uma máquina de busca e permite que usuários, gratuitamente, entrem com uma ou

mais palavras e recebam endereços da web associados às palavras. Apesar do serviço gratuito, a BUBBLE lucra apresentando propagandas de outras empresas que pagam para aparecer nas buscas relacionadas a elas. Para isto, a página de busca dispõe de uma região retangular na lateral da tela de busca. Tal região é estreita, de largura fixa, e preenche toda a altura da tela. Todas as propagandas devem ter a mesma largura, porém, podem ter alturas distintas, contanto que não passem da altura da região. Suponha que a busca realizada por um usuário define um conjunto de n propagandas $P = \{1, \dots, n\}$, associadas à sua busca. Cada propaganda $i \in P$, usa uma certa altura p_i da região e caso seja apresentada ao usuário, fornece um lucro de v_i reais para a BUBBLE. Sabendo que a altura da região é B , quais propagandas devem ser apresentadas de maneira a maximizar o valor recebido pela BUBBLE ?

Este é um típico problema de otimização combinatória. Os possíveis candidatos a solução são os subconjuntos das propagandas, mas são viáveis apenas aquelas que podem ser colocadas na região, sem ultrapassar sua altura. Naturalmente, procura-se uma que maximize o valor arrecadado. O problema de otimização subjacente é o Problema da Mochila Binária, definido formalmente a seguir.

Problema da Mochila Binária (PMB): Dados conjunto de itens $\{1, \dots, n\}$, cada item i com peso $p_i \geq 0$ e valor $v_i \geq 0$, ambos inteiros, e inteiro B , encontrar um conjunto $S \subseteq \{1, \dots, n\}$ tal que $\sum_{i \in S} p_i \leq B$ e $\sum_{i \in S} v_i$ é máximo.

De maneira geral, uma entrada I para um problema de otimização combinatória Π , deve ter um conjunto de possíveis soluções \mathcal{S}_I e uma função v , tal que $v(S)$ é o valor da solução S , para cada $S \in \mathcal{S}_I$. O objetivo é encontrar uma solução $S^* \in \mathcal{S}_I$ tal que $v(S^*)$ é máximo, se Π é de maximização, ou mínimo, se Π é de minimização. Neste caso, S^* é chamada *solução ótima de I*. Os elementos de \mathcal{S}_I (não necessariamente soluções ótimas) são chamados de *soluções viáveis*, ou simplesmente *soluções*. Qualquer elemento de \mathcal{S}_I poderia ser solução ótima de I , não o sendo apenas por existir outro elemento de \mathcal{S}_I com valor melhor para a função objetivo. Por vezes, define-se também um conjunto \mathcal{U}_I contendo os candidatos a soluções, com $\mathcal{S}_I \subseteq \mathcal{U}_I$, mas não necessariamente um elemento $U \in \mathcal{U}_I$ é uma solução viável de I . No exemplo da mochila binária, o conjunto \mathcal{U}_I poderia ser o conjunto de todos os subconjuntos de itens possíveis. Neste caso, nem todos os subconjuntos são possíveis de ser solução, uma vez que a soma dos pesos pode ser maior que a capacidade B da mochila. Os conjuntos em \mathcal{U}_I cuja soma dos pesos é no máximo B são as soluções viáveis de \mathcal{S}_I . Dentre estas, deve haver pelo menos uma solução que é ótima.

A seguir, será visto uma primeira técnica para resolver problemas de otimização combinatória.

3.2.1. Algoritmos Força-Bruta

Os algoritmos por força-bruta, são algoritmos que, como o nome diz, utilizam de muito esforço computacional para encontrar uma solução, sem se preocupar em explorar as

estruturas combinatórias do problema. De fato, estes algoritmos enumeram todas as possíveis soluções, verificando sua viabilidade e, caso satisfaça, seu potencial para ser uma solução ótima. Estão entre os algoritmos mais simples, porém, podem demandar grande quantidade de recursos computacionais sendo, no geral, consideradas apenas para entradas de pequeno porte.

Como exemplo, considere o Problema da Mochila Binária. Qualquer subconjunto de elementos é um potencial candidato a solução do problema, bastando para isso, que seu peso total não ultrapasse a capacidade da mochila. Assim, um algoritmo simples, poderia percorrer todos os subconjuntos possíveis e devolver, dentre aqueles que representam soluções viáveis, um de maior valor.

O algoritmo seguinte gera uma solução ótima para o problema da Mochila. As soluções são representadas por vetores binários, onde a i -ésima posição tem valor 1 se o item i pertence à solução e 0 caso contrário. O algoritmo é recursivo e além da entrada (B, p, v, n) , recebe outros dois parâmetros: x^* e x . O parâmetro x^* é um vetor binário n -dimensional que representa a melhor solução encontrada até o momento, e com isso, todas as referências a x^* são para a mesma variável. O parâmetro x é um vetor binário que a cada chamada recursiva, é acrescido de uma nova posição, que representa a pertinência de um item na solução. A primeira chamada do algoritmo é feita com os parâmetros $(B, p, v, n, x^*, ())$, onde x^* armazena a melhor solução obtida a cada momento, começando com o vetor **0**, que representa uma solução sem itens. O último parâmetro indica um vetor sem elementos. A base do algoritmo recursivo ocorre quando x contém n elementos binários, momento que representa um subconjunto dos itens da entrada.

Algoritmo: Mochila-FB(B, p, v, n, x^*, x)

Saída : Atualiza x^* se encontrar solução que completa x , e tem valor melhor

- 1 **se** $|x| = n$ **então**
 - 2 **se** $p \cdot x \leq B$ e $v \cdot x > v \cdot x^*$ **então** $x^* \leftarrow x$
 - 3 **senão**
 - 4 Mochila-FB($B, p, v, x^*, x \|(0)$)
 - 5 Mochila-FB($B, p, v, x^*, x \|(1)$)
-

No início de cada chamada, os elementos que já tiveram sua pertinência na atual solução definida, estão representados em x , faltando considerar os próximos $n - |x|$ itens. A execução deste algoritmo pode ser representada por uma árvore binária de enumeração, onde cada nó representa uma chamada recursiva e as arestas que ligam seus filhos, representam as duas situações possíveis, quando o item pertence à solução (ramo com $x_i = 1$) ou não (ramo com $x_i = 0$). Note que este algoritmo sempre gera uma árvore de enumeração completa.

No início de cada chamada recursiva, o Algoritmo Mochila-FB precisa considerar a pertinência dos próximos $n - |x|$ itens, sendo que x é acrescido de uma coordenada a cada chamada recursiva. O algoritmo pode ser implementado de maneira a ter complexidade de tempo limitada pela recorrência $T(n) = 2T(n - 1) + O(1)$, cuja resolução mostra que $T(n)$ é $O(2^n)$. Note que o passo 1 pode ser implementado em tempo constante, mantendo para cada vetor que representa uma solução, uma variável contendo o valor total dos itens atribuídos. Tal passo verificará todas os subconjuntos possíveis de itens, levando a um algoritmo bastante custoso na prática.

Um outro problema importante para a área de Otimização Combinatória é o Problema do Caixeiro Viajante. Trata-se de um problema clássico de grande importância, não apenas por suas várias aplicações, mas também por ser um problema para o qual diversas teorias foram desenvolvidas, testadas e difundidas.

Problema do Caixeiro Viajante (PCV): Dados um grafo completo $G = (V, E)$, cada aresta $e \in E$ com custo c_e , determinar um ciclo hamiltoniano C em G tal que $\sum_{e \in C} c_e$ é mínimo.

Um algoritmo força-bruta para o problema do caixeiro viajante poderia listar todas as sequências possíveis e armazenar a melhor. Como o grafo é completo, qualquer permutação dos vértices do grafo gera uma possível solução.

Algoritmo: Caixeiro-Viajante-FB(V, E, c)

Saída : Encontra um circuito gerador de custo mínimo

- 1 $\varphi^* \leftarrow \infty$
 - 2 **para** cada permutação (v_1, \dots, v_n) dos vértices de V **faça**
 - 3 $C \leftarrow \{\{v_i, v_{i+1}\} : i \in \{1, \dots, n-1\}\} \cup \{\{v_1, v_n\}\}$
 - 4 $\varphi \leftarrow \sum_{e \in C} c_e$
 - 5 **se** $\varphi < \varphi^*$ **então**
 - 6 $\varphi^* \leftarrow \varphi$
 - 7 $C^* \leftarrow C$
 - 8 **devolva** C^*, φ^*
-

Como o Algoritmo Caixeiro-Viajante-FB percorre por toda as permutações de vértices, faz claramente $n!$ iterações, para $n = |V|$. Demais operações em uma mesma iteração, são realizadas em tempo polinomial. Assim, a complexidade de tempo deste algoritmo é $\Omega(n!)$, e é assintoticamente maior que qualquer função do tipo a^n , para qualquer constante $a \geq 1$.

Para dar uma idéia do crescimento destas funções, considere um computador com velocidade de 1 Terahertz (mil vezes mais rápido que um computador de 1 Gigahertz) e funções que para cada tamanho n dão o número de instruções executadas

neste computador. A Tabela 3.1 mostra os tempos obtidos para algumas funções polinomiais e exponenciais, onde os tempos são dados em segundos (seg), dias e séculos (séc). Nota-se, para este exemplo, que o tempo computacional dado pelas funções exponenciais cresce muito mais rapidamente que as funções de tempo polinomial.

$f(n)!$	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
n	$2,0 \times 10^{-11}$ seg	$4,0 \times 10^{-11}$ seg	$6,0 \times 10^{-11}$ seg	$8,0 \times 10^{-11}$ seg	$1,0 \times 10^{-10}$ seg
n^2	$4,0 \times 10^{-10}$ seg	$1,6 \times 10^{-9}$ seg	$3,6 \times 10^{-9}$ seg	$6,4 \times 10^{-9}$ seg	$1,0 \times 10^{-8}$ seg
n^3	$8,0 \times 10^{-9}$ seg	$6,4 \times 10^{-8}$ seg	$2,2 \times 10^{-7}$ seg	$5,1 \times 10^{-7}$ seg	$1,0 \times 10^{-6}$ seg
2^n	$1,05 \times 10^{-6}$ seg	1,1seg	13,3 dias	383,1séc	$4,0 \times 10^8$ séc
3^n	$3,5 \times 10^{-3}$ seg	140,7 dias	$1,3 \times 10^7$ séc	$4,7 \times 10^{16}$ séc	$1,6 \times 10^{26}$ séc
$n!$	28,1dias	$2,6 \times 10^{26}$ séc	$2,6 \times 10^{60}$ séc	$2,3 \times 10^{97}$ séc	$2,9 \times 10^{136}$ séc

Tabela 3.1. Comparação de algumas funções de tempo computacional.

Os tempos computacionais indicados na Tabela 3.1 sugerem que algoritmos de tempo exponencial servirão apenas para entradas de tamanho muito pequeno e com utilidade bastante limitada. Por outro lado, dão também uma pequena amostra dos desafios encontrados na área de Otimização Combinatória. É uma área que contempla inúmeras aplicações práticas, onde mesmo pequenas melhorias algorítmicas podem representar uma economia de grande volume de recursos. Além disso, se torna envolvente na medida que mostra ser necessário explorar as estruturas combinatórias destes problemas de maneira a cercar as soluções ótimas ou mesmo as que apresentam proximidade em relação às soluções ótimas. Um típico exemplo é o Problema do Caixeiro Viajante, para o qual há algoritmos exatos com complexidade de tempo exponencial que encontraram soluções ótimas para entradas de 85900 vértices [Applegate et al. 2007]. Tal grandeza, confrontada com os tempos da Tabela 3.1, mostram claramente a importância de se investigar as estruturas combinatórias dos problemas na busca das melhores soluções.

3.3. Indução Matemática no Projeto de Algoritmos

A indução matemática é uma técnica poderosa de demonstração de resultados teóricos (teoremas, proposições, lemas, etc). O intuito desta seção é chamar a atenção para a analogia que existe entre o projeto de algoritmos e a prova de um resultado teórico usando indução matemática. Inicialmente, relembram-se os conceitos básicos de indução.

Uma prova por indução pode ser resumida do seguinte modo. Seja T um teorema que se quer demonstrar e suponha que o enunciado do resultado inclui um parâmetro n que é um número inteiro não-negativo. Genericamente, se Π é uma propriedade qualquer que depende de n , T pode ser enunciado por:

Para todo inteiro $n \geq n_0$, com $n_0 > 0$, a propriedade Π é verdadeira.

A técnica de prova por **indução fraca** para demonstrar T , se resume nos passos a seguir:

1. **base da indução:** mostrar que Π é verdadeira para n_0 ;
2. **hipótese de indução (fraca):** para $n > n_0$, supor que Π é verdadeira para $n - 1$;
3. **passo da indução:** a partir da hipótese de indução, provar que Π é verdadeira para n .

Note que a base garante a veracidade de Π para n_0 . A partir daí, o passo de indução mostra que a propriedade também é correta para $n_0 + 1$, já que a hipótese de indução vale para n_0 . Isso permite avançar um passo adiante pois, ao usar a validade de Π para $n_0 + 1$ no passo de indução, prova-se que Π é verdadeira para $n_0 + 2$. Fica claro que este argumento pode ser repetido indefinidamente, assegurando, então, que a propriedade vale para qualquer $n > n_0$. Alternativamente, a hipótese de indução pode ser alterada de modo a se exigir que ela valha para todo inteiro k tal que $n_0 \leq k \leq n - 1$. Caso isso seja feito, ela toma a forma abaixo:

- (2') **hipótese de indução (forte):** para $n > n_0$, supor que Π é verdadeira para todo $n_0 \leq k \leq n - 1$;

Essencialmente a estrutura da prova não muda, exceto que, no passo de indução, há uma possibilidade maior de uso da hipótese, uma vez que ela pode ser aplicada a diferentes valores de k . Considere o exemplo abaixo que, embora bastante simples, serve bem para ilustrar os conceitos.

Exemplo 3.3.1 *O serviço postal de um país hipotético tem selos de 4 e 5 centavos. Prove que qualquer valor de postagem n (em centavos) de uma carta, em que $n \geq 12$, pode ser conseguido usando-se apenas os selos de 4 e 5 centavos.*

Solução usando indução fraca:

- Base: o valor $n = 12$ pode ser conseguido usando-se 3 selos de 4 centavos.
- Hipótese: para $n > 12$, o valor de postagem $n - 1$ pode ser obtido usando-se p' selos de 4 e q' selos de 5 centavos.
- Passo: provar que existem valores p e q inteiros não-negativos, para os quais $n = 4p + 5q$, sendo $n > 12$. Primeiro, analisa-se o que ocorre quando $p' > 0$ na hipótese induktiva. Neste caso, faz-se $p = p' - 1$ e $q = q' + 1$, o que resulta em:

$$4p + 5q = 4(p' - 1) + 5(q' + 1) = 4p' + 5q' + 1 = (n - 1) + 1 = n.$$

No segundo caso, quando $p' = 0$, $n - 1$ deve ser múltiplo de 5. Como $n > 12$, tem-se que $n - 1 \geq 15$ e, portanto, $q' \geq 3$. Logo, fazendo-se $p = 4$ e $q = q' - 3$, tem-se que

$$4p + 5q = 4 \times 4 + 5(q' - 3) = 5q' + 1 = (n - 1) + 1 = n. \quad \square$$

Solução usando indução forte:

- Base: para $n = 12$ o valor pode ser conseguido usando-se 3 selos de 4 centavos. Nada impede que o resultado possa ser mostrado para outros valores pequenos de n . Isso é feito aqui até $n = 15$ e as razões para tal são explicadas mais abaixo. Assim, tem-se que: $n = 13 = 2 \times 4 + 1 \times 5$, $n = 14 = 1 \times 4 + 2 \times 5$ e $n = 15 = 0 \times 4 + 3 \times 5$.
- Hipótese: para $n > 15$, qualquer valor de postagem k , com $15 \leq k \leq n - 1$ pode ser obtido usando-se selos de 4 e de 5 centavos.
- Passo: provar que existem valores p e q inteiros não-negativos, para os quais $n = 4p + 5q$, sendo $n > 15$. Como $n > 15$, $n - 4 \geq 12$. Nesta situação, a hipótese de indução garante que existem inteiros não-negativos p' e q' tais que $n - 4 = 4p' + 5q'$. Logo, se $p = p' + 1$ e $q' = q$, tem-se

$$4p + 5q = 4(p' + 1) + 5q' = 4p' + 5q' + 4 = (n - 4) + 4 = n. \quad \square$$

Na prova por indução forte, pode-se perguntar por que a demonstração da base se estendeu até $n = 15$. É muito importante perceber que isto ocorreu porque no passo de indução foi usado o fato de que o valor de postagem $n - 4$ pode ser pago usando apenas os selos de 4 e 5 centavos. Se na base não tivessem sido mostrados os casos para $n = 13, 14$ e 15 , o passo de indução não teria provado o resultado para os valores de postagem, $17, 18$ e 19 ! Com isto, não estaria demonstrada a veracidade da proposição para os valores $\{21, 22, 23, 25, 26, 27, \dots\}$. Portanto, é preciso estar atento para que a base da indução (ou seja, a prova da validade da proposição para pequenos valores de n) seja feita até um valor grande o suficiente para poder ser usado na prova do passo.

Uma vez repassados os conceitos preliminares de indução matemática, examina-se a seguir como usar a técnica no projeto de algoritmos. De acordo com [Manber 1989], o princípio básico ao desenvolver um algoritmo para um dado problema pode ser resumido assim. *Não é preciso projetar todos os passos requeridos para resolver o problema “do zero”. Na verdade, é suficiente garantir que (a) sabe-se como resolver pequenas instâncias do problema (caso base), e (b) a solução para a instância original do problema pode ser obtida a partir da solução de instâncias menores do mesmo problema (passo indutivo).* Como será visto, a prova por indução de que é sabido como resolver o problema para uma instância de tamanho, por exemplo, n , dá origem de modo bastante natural a um algoritmo recursivo. As chamadas recursivas são feitas para obter as soluções das instâncias de menor tamanho, enquanto o caso base da recursão confunde-se com o caso base da indução. O passo de indução diz como, a partir das soluções das instâncias menores, pode ser feito o cálculo que leva à solução da instância original. Três observações são importantes neste processo. Primeiramente, perceba que o fato do algoritmo obtido da prova por indução ser recursivo em nada restringe a técnica, até porque, sabe-se que todo algoritmo recursivo pode ser implementado de forma iterativa,

muitas vezes sem a necessidade de uso de pilhas. A segunda observação diz respeito à análise da complexidade do algoritmo obtido a partir da prova por indução a qual, pela natureza recursiva do algoritmo, é expressa por uma fórmula de recorrência. Finalmente, destaca-se que, como foi visto, há duas formas de prova por indução dependendo se a hipótese indutiva usada for a fraca ou a forte. Isso se reflete no projeto do algoritmo pois, para a indução fraca, ele seguirá o *paradigma incremental*, enquanto que, para a indução forte, ele seguirá o *paradigma de divisão e conquista*.

Para ilustrar as ideias discutidas no parágrafo anterior, aplica-se a técnica de projeto por indução ao *problema do subvetor de soma máxima* (PSSM) descrito a seguir.

Problema do Subvetor de Soma Máxima (PSSM): Dado vetor $A[1, \dots, n]$ de inteiros irrestritos em sinal. encontrar o valor correspondente à maior soma de elementos que pode ter um subvetor de A .

Antes de prosseguir, é importante lembrar que, para todo $1 \leq i \leq j \leq n$, tem-se que $A[i, \dots, j]$ é um subvetor de A , cuja soma dos elementos será denotada por $s(i, j)$, i.e., $s(i, j) = \sum_{k=i}^j A[k]$. Consequentemente, o que se quer é encontrar o valor $s(i^*, j^*)$, onde o par (i^*, j^*) é tal que $s(i^*, j^*) = \max_{1 \leq i \leq j \leq n} \{s(i, j)\}$. Por convenção, se um vetor só possui elementos negativos, a soma máxima tem valor nulo.

Claro que este problema admite uma solução ingênuo em que se calcula o valor de $s(i, j)$ para todos pares (i, j) escolhidos de modo que $1 \leq i \leq j \leq n$. É fácil ver que a complexidade deste algoritmo é $O(n^3)$. Como será visto, o projeto de um algoritmo para o PSSM tanto pelo paradigma incremental (indução fraca) quanto pelo paradigma de divisão e conquista (indução forte) irá resultar em um algoritmo de complexidade menor que a do algoritmo ingênuo. Em ambos os casos, a princípio, a proposição a ser mostrada é a seguinte:

Sabe-se resolver o PSSM para todo vetor de tamanho $n \geq 1$.

Evidentemente, a indução será feita em n . Contudo, uma tentativa direta de prova desta proposição não conduz a um algoritmo com complexidade menor do que aquela do algoritmo ingênuo. Para conseguir alguma melhoria, deve-se *reforçar a hipótese*, ou seja, exigir que além de resolver o PSSM, devolvendo a sua solução, outras informações possam ser devolvidas pelo algoritmo também. O *reforço da hipótese* é uma técnica muito comum em provas por indução. Embora em um primeiro momento possa parecer contra-intuitivo que, ao se exigir provar um resultado mais forte a prova acabe se simplificando, isso pode ocorrer devido à natureza da prova por indução. Repare que a hipótese reforçada é carregada para o passo da indução, fazendo com que mais informações possam ser usadas neste último. Tendo este fato em mente, seguem-se os exemplos.

Exemplo 3.3.2 (Projeto de um algoritmo incremental para o PSSM)

Considere a seguinte proposição já com a hipótese reforçada:

Sabe-se resolver o PSSM para todo vetor de tamanho $n \geq 1$ e também calcular o valor da maior soma de um sufixo deste vetor.

O sufixo (prefixo) de um vetor é um subvetor que termina (começa) na última (primeira) posição do mesmo. O valor a ser devolvido pelo PSSM será denotado por v e o valor do maior sufixo por s . Feitos estes esclarecimentos, a prova seria a seguinte.

Base: para $n = 1$, se $A[1] \geq 0$, $v = s = A[1]$, caso contrário, $v = s = 0$.

Hipótese (indução fraca): Para $n > 1$, sabe-se resolver o PSSM para todo vetor de tamanho $n - 1$ e também calcular o valor da maior soma de um sufixo deste vetor.

Passo: mostra-se que a proposição é verdadeira para n usando a hipótese. Para tanto, sejam v' e s' os valores devolvidos pela aplicação da hipótese ao prefixo de tamanho $n - 1$ do vetor A dado na entrada. Neste caso, os cálculos de v e s são simples. Primeiramente, v é computado por $v = \max\{v', s' + A[n]\}$. Isto ocorre porque, ou o subvetor que corresponde à solução ótima do prefixo $A[1, \dots, n - 1]$ continua sendo solução ótima para A e, portanto, não é sufixo de A , ou o sufixo de maior valor terminando em $A[n]$ é que fornece a solução ótima de A . Já o valor de s é dado por $\max\{s' + A[n], 0\}$ já que, o melhor sufixo ou é estendido com a inclusão de $A[n]$, ficando com soma não negativa, ou a inclusão deste elemento torna a soma negativa, fazendo com que o melhor sufixo seja o subvetor vazio, cuja soma é nula por definição. \square

O pseudocódigo do algoritmo recursivo que resulta da prova por indução fraca é mostrado no Algoritmo PSSM. A complexidade deste algoritmo é dada pela fórmula de recorrência

$$T(n) = T(n - 1) + O(1),$$

e, portanto, $T(n) \in O(n)$, bem mais eficiente do que o algoritmo ingênuo.

Exemplo 3.3.3 (Projeto de um algoritmo de divisão e conquista para o PSSM)

Considere a seguinte proposição já com a hipótese reforçada:

Sabe-se resolver o PSSM para todo vetor de tamanho $n \leq 1$ e também calcular (i) o valor da maior soma de um sufixo deste vetor, (ii) o valor da maior soma de um prefixo deste vetor, e (iii) o valor da soma de todos os elementos do vetor.

O valor a ser devolvido pelo PSSM será denotado por v , o valor do maior sufixo (prefixo) por s (p) e a soma dos elementos por S .

Base: para $n = 1$, se $A[1] \geq 0$, $v = s = p = A[1]$, caso contrário, $v = s = p = 0$. Em qualquer caso, $S = A[1]$.

Algoritmo: PSSM(A, n, v, s) - Algoritmo incremental para o PSSM

```

Saída : Valor do maior sufixo,  $v$ , e prefixo  $s$ .
1 se  $n = 1$  então
2    $\triangleright$  caso base
3   se  $A[1] \geq 0$  então  $v \leftarrow s \leftarrow A[1]$ 
4   senão  $v \leftarrow s \leftarrow 0$ 
5 senão
6   PSSM( $A, n - 1, v', s'$ )  $\triangleright$  hipótese indutiva
7    $\triangleright$  passo de indução
8    $v \leftarrow \max\{v', s' + A[n]\}$ 
9    $s \leftarrow \max\{s' + A[n], 0\}$ 
10 devolva ( $v, s$ )

```

Hipótese (indução forte): Para $n > 1$, sabe-se resolver o PSSM para todo vetor de tamanho k , para qualquer k satisfazendo $1 \leq k \leq n - 1$ e também calcular o valor da maior soma de um sufixo deste vetor, assim como a de um prefixo e a soma total dos seus elementos.

Passo: mostra-se que a proposição é verdadeira para n usando a hipótese. Seja $m = \lfloor \frac{n}{2} \rfloor$. Suponha, pela hipótese indutiva, que o PSSM foi resolvido para o prefixo $A[1, \dots, m]$, devolvendo os valores v_e , s_e , p_e e S_e correspondentes às maiores somas de um subvetor qualquer, de um sufixo e de um prefixo, além da soma total dos seus elementos, respectivamente. Da mesma maneira, a hipótese indutiva aplicada ao sufixo $A[m+1, \dots, n]$ devolve os valores v_d , s_d , p_d e S_d . A partir das informações devolvidas pelos dois subproblemas, os valores respectivos de v , s , p e S a serem devolvidos pelo PSSM quando aplicado a A podem ser computados conforme explicado a seguir.

Obviamente, tem-se que $S = S_e + S_d$. É fácil ver que o prefixo de maior soma de A ou é o mesmo de $A[1, \dots, m]$ ou contém todo este subvetor e mais o maior prefixo de $A[m+1, \dots, n]$. Consequentemente, chega-se a $p = \max\{p_e, S_e + p_d\}$. Usando argumentos simétricos para o sufixo, pode-se concluir que $s = \max\{s_d, S_d + s_e\}$. Finalmente, para o cálculo de v , observa-se que o subvetor de maior soma em A só pode ser: (a) o mesmo de $A[1, \dots, m]$ ou (b) o mesmo de $A[m+1, \dots, n]$ ou (c) um subvetor que começa em uma posição que está em $A[1, \dots, m]$ e termina em uma posição que está em $A[m+1, \dots, n]$. Caso a condição (c) seja a que prevalece, percebe-se que o subvetor de soma máxima está dividido em duas partes, uma que é sufixo de $A[1, \dots, m]$ e outra que é prefixo de $A[m+1, \dots, n]$. Porém, a maior soma atingida por um subvetor nesta situação tem que corresponder à soma do valor máximo de um sufixo de $A[1, \dots, m]$ com o valor máximo de um prefixo de $A[m+1, \dots, n]$. Em outras palavras, na condição (c) acima, $v = s_e + p_d$. Assim, no caso geral, tem-se que $v = \max\{v_e, v_d, s_e + p_d\}$. \square

O pseudocódigo do algoritmo recursivo que resulta da prova por indução forte é mostrado na Figura 3.1 Resta analisar a complexidade do algoritmo de divisão e conquista para o PSSM.

```

1. PSSM-DC( $A, i, j, v, s, p, S$ )
(* resolve o PSSM para o subvetor  $A[i, \dots, j]$  de  $A$  *)
(*  $v, s, p$  e  $S$  são devolvidos pelo algoritmo *)
2. se  $i = j$  então (* caso base *)
3.     se  $A[i] \geq 0$ , então  $v = s = p = A[i]$ 
4.     se não  $v = s = p = 0$ 
5.      $S \leftarrow A[i]$ 
6. se não
7.      $m \leftarrow (i + j)/2$  (*  $m$ : posição mediana *)
8.     PSSM-DC( $A, i, m, v_e, s_e, p_e, S_e$ )      (* hipótese indutiva *)
9.     PSSM-DC( $A, m + 1, j, v_d, s_d, p_d, S_d$ ) (* hipótese indutiva *)
(* passo de indução *)
10.     $v \leftarrow \max\{v_e, v_d, s_e + p_d\}$ 
11.     $s \leftarrow \max\{s_d, S_d + s_e\}$ 
12.     $p \leftarrow \max\{p_e, S_e + p_d\}$ 
13.     $S \leftarrow S_e + S_d$ 
14. fim.

```

Figura 3.1. Algoritmo de divisão e conquista para o PSSM.

quista para o PSSM. Aproveita-se esta oportunidade para apresentar o chamado *Teorema Mestre*, que é muito útil no cálculo de equações de recorrência como as que representam a complexidade de algoritmos projetados através da estratégia de divisão e conquista.

Suponha que se tenha um algoritmo de divisão e conquista para um problema cuja entrada é medida pelo parâmetro n , um número inteiro não negativo. Tirando os casos *base* correspondentes a instâncias pequenas em que o problema é resolvido de forma simples e em tempo $O(1)$, para duas constantes a e b , $a \geq 1$ e $b > 1$, o algoritmo encontra a solução da instância original subdividindo-a em a instâncias menores de tamanho aproximadamente n/b que são resolvidas recursivamente. Ao final, as soluções dos a subproblemas são combinadas por um algoritmo de complexidade $f(n)$ que calcula a solução da instância original. Então, se $T(n)$ denota a complexidade do algoritmo de divisão e conquista, tem-se que $T(n) = aT(n/b) + f(n)$ para todo $n \geq n_0$ para algum $n_0 > 0$. Neste contexto, o resultado abaixo é passível de ser aplicado.

Teorema 3.3.1 (Teorema Mestre (cf., [Cormen et al. 2009])) *Sejam $a \geq 1$ e $b \geq 2$ constantes, $f(n)$ uma função e $T(n)$ definida para os inteiros não-negativos pela relação de recorrência*

$$T(n) = aT(n/b) + f(n).$$

Então $T(n)$ pode ser limitada assintoticamente da seguinte maneira:

1. Se $f(n) \in O(n^{\log_b a - \varepsilon})$ para alguma constante $\varepsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$.
2. Se $f(n) \in \Theta(n^{\log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log n)$.
3. Se $f(n) \in \Omega(n^{\log_b a + \varepsilon})$, para algum $\varepsilon > 0$ e se $a f(n/b) \leq c f(n)$, para alguma constante $c < 1$ e para n suficientemente grande, então $T(n) \in \Theta(f(n))$.

Neste teorema, o valor de n/b na fórmula de recorrência pode ser substituído indistintamente por $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$ em qualquer uma das a parcelas, sem que o resultado final seja alterado.

Assim, para o algoritmo de divisão e conquista desenvolvido acima para o PSSM, tem-se que $T(n) = 2T(n/2) + O(1)$, fazendo com que $a = 2$, $b = 2$ e $f(n) \in O(1)$ (onde $T(n/2)$ é a complexidade de resolver o PSSM em cada uma das suas metades). Desta forma, $\log_b a = 1$ e, assim, chega-se a $f(n) \in O(n^{\log_b a - \varepsilon})$, para qualquer ε no intervalo $(0, 1)$. Conclusão: a recorrência acima cai no caso 1 do Teorema Mestre, o que implica que $T(n) = \Theta(n)$. Ou seja, a complexidade assintótica do algoritmo de divisão e conquista é idêntica àquela do algoritmo incremental.

Na análise da complexidade de tempo de alguns algoritmos apresentados no texto, será necessário resolver recorrências lineares homogêneas e não-homogêneas. As recorrências homogêneas são equações lineares onde um termo da recorrência é definido a partir de outros valores da própria recorrência, possivelmente multiplicadas por constantes, mas não contém termos sem referência à recorrência. Uma recorrência homogênea f_n definida através de k termos anteriores, têm a seguinte forma:

$$c_n f_n + c_{n-1} f_{n-1} + \cdots + c_{n-k} f_{n-k} = 0,$$

onde f_i é a recorrência para um valor i e c_i são termos que independem da recorrência, com c_n e c_{n-k} não-nulos. A equação característica desta recorrência é dada por

$$c_k x^k + c_{k-1} x^{k-1} + \cdots + c_0 = 0.$$

Como a recorrência acima possui $k+1$ termos, é necessário que a recorrência seja definida também com os k primeiros termos de maneira direta, para que os próximos termos possam ser obtidos através da relação de recorrência. Para a obtenção de uma cota superior, pode-se utilizar o seguinte teorema.

Teorema 3.3.1 (cf. [Brassard e Bratley 1988, Rosen 2012]) *Seja f_n uma recorrência linear homogênea e $p(n)$ sua equação característica. Se $p(n)$ tem raízes a_1, \dots, a_r , cada raiz a_i com multiplicidade m_i , então f_n é dada por*

$$f_n = \sum_{i=1}^r \sum_{j=0}^{m_i-1} c_{i,j} n^j a_i^n,$$

onde os termos $c_{i,j}$ são constantes.

Exemplo 3.3.4 Considere a recorrência de Fibonacci, $F(n) = F(n-1) + F(n-2)$, para $n \geq 2$. Reorganizando a recorrência, temos $F(n) - F(n-1) - F(n-2) = 0$. A equação característica é dada por $a^2 - a - 1 = 0$ cujas soluções são $a_1 = (1 + \sqrt{5})/2$ e $a_2 = (1 - \sqrt{5})/2$, ambas com multiplicidade $m_1 = m_2 = 1$. Assim, a recorrência $F(n)$ é dada por $F(n) = c_1 n^0 a_1^n + c_2 n^0 a_2^n = c_1 a_1^n + c_2 a_2^n$. De maneira assintótica temos que $F(n)$ é $O\left(\frac{1+\sqrt{5}}{2}\right)^n$. De maneira mais precisa, podemos utilizar os valores de $F(0)$ e $F(1)$ para determinar os valores de c_1 e c_2 , pelo sistema de equações

$$\begin{aligned} F(0) &= c_1 + c_2 = 0 \\ F(1) &= c_1 \frac{1+\sqrt{5}}{2} + c_2 \frac{1-\sqrt{5}}{2} = 1, \end{aligned}$$

que resolvendo, nos dá $c_1 = 1/\sqrt{5}$ e $c_2 = -1/\sqrt{5}$. Portanto, os números de Fibonacci são dados por

$$F(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n.$$

Para se obter limitantes para as recorrências não-homogêneas, pode-se utilizar o seguinte teorema.

Teorema 3.3.2 (cf. [Brassard e Bratley 1988]) Seja f_n uma recorrência linear não-homogênea dada por

$$c_k f_n + c_{k-1} f_{n-1} + \cdots + c_0 f_{n-k} = b_1^n q_1(n) + b_2^n q_2(n) + \cdots + b_t^n q_t(n), \quad (1)$$

onde (i) c_0, \dots, c_k são constantes; (ii) b_1, \dots, b_t são constantes distintas e q_1, \dots, q_t são polinômios de grau d_1, \dots, d_t , respectivamente. O polinômio característico desta recorrência é dado por

$$p(n) = (c_k x^k + c_{k-1} x^{k-1} + \cdots + c_0)(x - b_1)^{d_1+1}(x - b_2)^{d_2+1} \cdots (x - b_t)^{d_t+1}.$$

Se $p(n)$ tem r raízes distintas, a_1, \dots, a_r , cada raiz a_i com multiplicidade m_i , então f_n é dada por

$$f_n = \sum_{i=1}^r \sum_{j=0}^{m_i-1} e_{i,j} n^j a_i^n,$$

onde os termos $e_{i,j}$ são constantes.

Note que o polinômio característico $p(n)$ possui uma parte que representa o polinômio característico da recorrência homogênea, obtida removendo os termos do lado direito da recorrência (1), e outra advinda dos termos do lado direito da recorrência.

O próximo exemplo é apresentado em [Brassard e Bratley 1988].

Exemplo 3.3.5 Vamos resolver a recorrência $t_n = 2t_{n-1} + n + 2^n$ quando $n \geq 1$ e $t_0 = 0$. Podemos reescrevê-la como

$$t_n - 2t_{n-1} = n + 2^n. \quad (2)$$

A recorrência homogênea obtida só pelo lado esquerdo de (2) é dada por $t_n - 2t_{n-1} = 0$, que possui equação característica $x - 2 = 0$, que claramente tem raiz 2.

Considere o lado direito de (2). Vamos colocar na forma requerida pelo Teorema 3.3.2. Podemos escrever $n + 2^n$ como

$$\begin{aligned} n + 2^n &= 1^n n + 2^n 1 \\ &= b_1^n p_1(n) + b_2^n p_2(n), \end{aligned}$$

onde $b_1 = 1$, $p_1(n) = n$, $b_2 = 2$ e $p_2(n) = 1$. O grau do polinômio $p_1(n)$ é $d_1 = 1$ e de $p_2(n)$ é $d_2 = 0$, temos a seguinte equação característica para a recorrência não-homogênea: $(x - 2)(x - 1)^{1+1}(x - 2)^{0+1} = 0$, que simplificando, nos dá

$$(x - 2)^2(x - 1)^2 = 0.$$

Como temos duas raízes com valor 1 e duas raízes com valor 2, ambas com multiplicidade 2, podemos escrever a recorrência na forma

$$\begin{aligned} t_n &= c_1 1^n + c_2 n 1^n + c_3 2^n + c_4 n 2^n \\ &= c_1 + c_2 n + c_3 2^n + c_4 n 2^n. \end{aligned}$$

Com isso, já podemos concluir que t_n é $O(n2^n)$. Para calcular a recorrência exata, devemos calcular os valores de c_1, c_2, c_3 e c_4 , que podem ser obtidos resolvendo-se um sistema obtido com os quatro primeiros valores da recorrência: t_0, \dots, t_3 .

$$\begin{array}{lllll} c_1 + & c_3 & = & 0 & \text{para } n = 0 \\ c_1 + & c_2 + & 2c_3 + & 2c_4 & = 3 \quad \text{para } n = 1 \\ c_1 + & 2c_2 + & 4c_3 + & 8c_4 & = 12 \quad \text{para } n = 2 \\ c_1 + & 3c_2 + & 8c_3 + & 24c_4 & = 35 \quad \text{para } n = 4. \end{array}$$

Tal sistema nos dá $c_1 = -2$, $c_2 = -1$, $c_3 = 2$ e $c_4 = 1$. Portanto, a recorrência é dada por

$$t_n = -2 - n + 2^{n+1} + n 2^n.$$

Para mais detalhes sobre recorrências, veja [Brassard e Bratley 1988, Manber 1989] e [Cormen et al. 2009].

3.4. Algoritmos Gulosos

O projeto por algoritmos gulosos está, em geral, associado a técnicas simples, intuitivas e, em geral, fáceis de implementar. Sendo assim, uma das técnicas mais utilizadas no projeto de algoritmos para problemas de otimização combinatória. Para aplicar esta técnica, a solução deve ser construída a partir de elementos básicos. Em um problema de minimização, cada elemento possui um custo e sua escolha fornece uma contribuição para a construção de uma solução. Analogamente, em um problema de maximização, cada elemento gera um benefício, porém, consome parte dos recursos disponíveis. A idéia básica do projeto de um algoritmo guloso é, iterativamente selecionar um elemento que melhor contribua para a construção de uma solução, considerando seu custo ou benefício. Elementos cuja escolha não levam a uma solução viável são descartados. O algoritmo termina quando os elementos selecionados formarem uma solução ou contiverem uma solução, que pode ser extraída de maneira direta.

A técnica é exemplificada com os problemas da Árvore Geradora de Custo Mínimo, do Caminho Mínimo e da Mochila Fracionária.

3.4.1. Algoritmo Guloso para o Problema da Árvore Geradora de Custo Mínimo

Esta seção trata do problema da Árvore Geradora de Custo Mínimo, que tem várias aplicações em estruturas que envolvem conectividade. Exemplos contemplam os projetos de redes de computadores, de circuitos VLSI, de tubulações, de saneamento, de telecomunicações, etc. Além disso, o problema aparece como subproblema de vários problemas de otimização.

Problema da Árvore Geradora de Custo Mínimo (PAGM): Dado grafo conexo, não-orientado $G = (V, E)$, cada aresta $e \in E$ com custo c_e , encontrar uma árvore geradora T de G , tal que $\sum_{e \in T} c_e$ é mínimo.

Um dos algoritmos que resolvem este problema na otimalidade, de maneira elegante, é o Algoritmo de Prim. A estratégia usada por este algoritmo, é começar uma árvore a partir de um vértice qualquer e a cada iteração uma aresta é acrescida, conectando a árvore corrente com um novo vértice.

A escolha de cada aresta é feita utilizando a estratégia gulosa. Isto é, dentre as arestas que acrescentam mais um vértice à árvore, é escolhida uma de custo mínimo. Interpretando o custo de uma aresta como distância, o algoritmo iterativamente anexa um vértice que está mais próximo da árvore. O algoritmo termina quando todos os vértices forem anexados à árvore. A seguir, as iterações são descritas de maneira mais precisa.

Denote por T_i a árvore construída até o início da i -ésima iteração e Q_i os vértices que não pertencem a T_i . No decorrer da iteração, os vértices de T_i são denominados internos e os de Q_i de externos. As arestas que ligam vértices de T_i a vértices de Q_i são chamadas arestas do corte. Dentre estas, usando estratégia gulosa, a aresta escolhida para fazer parte da solução na i -ésima iteração é uma aresta do corte que tem custo

mínimo.

Para que a escolha da aresta do corte possa ser feita rapidamente, o algoritmo usa vetores $\text{pred}[\cdot]$ e $\text{dist}[\cdot]$ para representar as arestas da árvore e agilizar sua busca. Se $u \in Q_i$ então $\text{dist}[u]$ contém o custo de uma aresta (do corte) que liga u a algum vértice $w \in T_i$, e neste caso guarda tal ligação fazendo $\text{pred}[u] = w$. Note que a representação das arestas definidas por pred , definem uma árvore enraizada no primeiro vértice da árvore. Com isso, a busca pela aresta do corte de custo mínimo recai na busca de um vértice $u \in Q_i$ com $\text{dist}[u]$ mínimo. Para isto, usa-se uma fila de prioridade em dist , que permite remover um vértice com menor $\text{dist}[u]$ e diminuir o valor de $\text{dist}[u]$ quando for encontrada aresta mais leve ligando u à T_i .

Na primeira iteração a árvore T_1 começa vazia. A cada iteração, o vértice externo a ser anexado, possui menor $\text{dist}[u]$. Note que na primeira iteração todos os vértices começam com $\text{dist}[v] = \infty$, exceto pela raiz r que começa com $\text{dist}[r] = 0$, sendo portanto o primeiro vértice inserido em T_1 . Após incluir um vértice u em T_i , os valores de $\text{dist}[v]$ são atualizados para os vértices v que são externos e adjacentes a u , já que as arestas adjacentes a u serão as únicas da iteração a sair ou entrar no corte. Após esta operação, os vetores $\text{pred}[\cdot]$ e $\text{dist}[\cdot]$ podem ficar incorretos para os vértices de Q_i adjacentes a u . Assim, para cada aresta do corte que liga u a um vértice $v \in Q_i$, atualizamos, se necessário, $\text{pred}[v]$ e $\text{dist}[v]$.

Algoritmo: $\text{Prim}(G, r)$ onde $G = (V, E, c)$ é um grafo ponderado e $r \in V$.

Saída : Árvore geradora de custo mínimo.

```

1 para  $v \in V$  faça
2    $\text{dist}[v] \leftarrow \infty$ 
3    $\text{pred}[v] \leftarrow \text{nil}$ 
4    $\text{dist}[r] \leftarrow 0$ 
5    $Q \leftarrow V$ 
6 enquanto  $Q \neq \emptyset$  faça
7    $u \leftarrow \text{Extrai-Min}(Q)$ 
8   para cada  $v \in \text{Adj}(u) \cap Q$  faça
9     se  $c(u, v) < \text{dist}[v]$  então
10        $\text{pred}[v] \leftarrow u$ 
11        $\text{dist}[v] \leftarrow c(u, v)$ 
12 devolva ( $\text{pred}$ )

```

Lema 3.4.1 Seja T_{i-1} uma árvore que pode ser estendida para uma árvore geradora de custo mínimo. Se e é uma aresta de custo mínimo com exatamente uma extremidade em T_{i-1} , então, a árvore $T_i = T_{i-1} + e$, pode ser estendida para uma árvore geradora de custo mínimo de G .

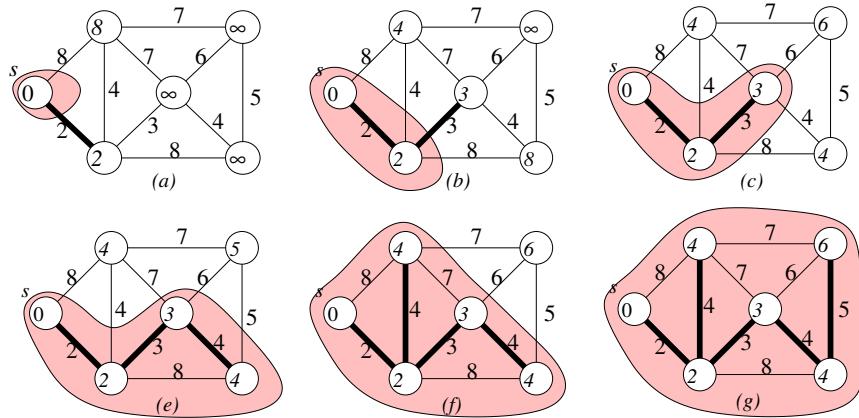


Figura 3.2. Simulação da execução do Algoritmo de Prim.

Prova. Seja T_{i-1} uma árvore que pode ser estendida para uma solução ótima. Seja T^* uma árvore geradora de custo mínimo que contém T_{i-1} .

A árvore $T^* + e$ possui um circuito passando por e . Como e liga um vértice de T_{i-1} a um vértice fora de T_{i-1} , existe uma aresta f do circuito que também tem exatamente uma extremidade em T_{i-1} . Portanto, a árvore $T^+ = T^* + e - f$ é uma árvore geradora. Pela escolha gulosa da aresta e vale que $c(e) \leq c(f)$, e portanto $c(T^+) \leq c(T^*)$. Como T^* é de custo mínimo, temos que T^+ também é de custo mínimo e contém a árvore T_i . \square

Teorema 3.4.2 *Para todo grafo ponderado e conexo, o Algoritmo de Prim encontra uma árvore de custo mínimo.*

Prova. Considere a seguinte afirmação:

No fim da iteração i , as seguintes propriedades são válidas, para $i = 1, \dots, n$.

- Se T_i é a árvore obtida no fim da iteração i pelo Algoritmo de Prim, então T_i pode ser estendida para uma árvore geradora de custo mínimo.
- Para todo vértice $v \notin T_i$, tem-se que $\text{dist}[v]$ é o menor custo de uma aresta que liga v à algum vértice de T_i , através de uma aresta $\{v, \text{pred}[v]\}$.

A prova será por indução no número da iteração i , para $i = 1, \dots, n$.

Base: para $i = 1$, a árvore T_1 , obtida ao final da primeira iteração, é a árvore contendo apenas o vértice raiz, que está contido em qualquer árvore geradora de custo mínimo. Ao inserir o vértice raiz, as únicas arestas necessárias para atualização são as arestas bincidentes à raiz.

Hipótese (indução fraca): No fim da iteração i , temos uma árvore T_i que pode ser estendida para uma árvore geradora de custo mínimo. Os vetores pred e dist indicam uma aresta de menor custo que liga $w \notin T_i$ à algum vértice de T_i . O custo desta aresta é dado por $\text{dist}[w]$.

Passo: considere uma árvore T_i obtida ao fim da iteração i . Esta é a mesma árvore obtida no início da iteração $i+1$. Seja u o vértice escolhido pelo algoritmo. Pela hipótese de indução, temos que $\{u, \text{pred}[u]\}$ é uma aresta de custo mínimo com exatamente uma extremidade em T_i . Ao incluir o vértice u , o custo de uma aresta com exatamente uma extremidade em T_{i+1} pode alterar apenas se for adjacente a u . Assim, basta atualizar, se necessário, as distâncias para os vértices adjacentes à u com vértices fora de T_{i+1} . Como a aresta $\{u, \text{pred}[u]\}$ tem custo mínimo, dentre as arestas que ligam T_i a Q_i , temos pelo Lema 3.4.1 que existe árvore geradora de custo mínimo que contém T_{i+1} . \square

Nem sempre é possível garantir que um algoritmo guloso produza uma solução ótima para um problema. Mas quando o problema admite certas propriedades, é possível que tais algoritmos produzam soluções ótimas. Duas propriedades que costumam aparecer no projeto de algoritmos gulosos que produzem soluções ótimas, são a da subestrutura ótima, que costuma aparecer nos problemas que podem ser resolvidos por projeto de indução, e da escolha gulosa.

Na resolução de um problema pelo projeto por indução, definimos o problema de maneira indutiva, ou um outro cuja resolução nos leve a uma solução ótima do problema original. Para se ter a propriedade de subestrutura ótima, uma solução ótima de um problema deve conter soluções ótimas para seus subproblemas. Note que o problema PAGM admite subestrutura ótima. O problema poderia ser definido com a seguinte subestrutura: Dado grafo G e subárvore T de G , encontrar uma árvore geradora de G de custo mínimo que contém T . A outra propriedade é a da escolha gulosa. Esta propriedade diz que ao escolher um próximo elemento a de maneira gulosa, ainda temos condições de atingir uma solução ótima. De fato, esta foi a propriedade usada no Lema 3.4.1, que garante que a cada iteração fazemos uma escolha gulosa que pertencerá a uma árvore geradora de custo mínimo.

A complexidade de tempo do Algoritmo de Prim depende da implementação de Q . Abaixo, temos a análise por duas estruturas de dados, o heap binário e o heap de Fibonacci. Para mais informações sobre estas estruturas de dados, veja [Cormen et al. 2009, Manber 1989]. Se for usado um heap binário, pode-se construir o heap da linha 5 em tempo $O(|V|)$. A cada iteração do laço do passo 6 é removido um elemento de Q . Como o laço é iterado por $|V|$ vezes e cada chamada da rotina $\text{Extrai-Min}(Q)$ é feita em tempo $O(\log(|V|))$, o passo 7 é executado em $O(|V| \log |V|)$. O laço do passo 6 juntamente com o laço do passo 8 percorre todas as arestas do grafo, visitando cada uma duas vezes, uma para cada extremidade. Portanto os passos 8–10 são executados $O(|E|)$ vezes. Já o passo 11 é uma operação de decrementação, que em um heap binário pode ser implementado com tempo $O(\log |V|)$. Portanto o tempo total desta implementação

é $O(|V| \log |V| + |E| \log |V|)$, i.e., $O(|E| \log |V|)$.

No caso de se usar um heap de Fibonacci, a operação Extrai-Min pode ser feita em tempo amortizado $O(\log V)$ e a operação que decrementa um valor do heap pode ser feita em tempo amortizado $O(1)$. Assim, esta implementação tem complexidade de tempo $O(|E| + |V| \log |V|)$.

3.4.2. Algoritmo Guloso para o Problema dos Caminhos Mínimos

Esta seção aborda o problema de se encontrar caminhos de custo mínimo de um vértice aos demais vértices em um grafo orientado com custos não-negativos nos arcos. Trata-se de um dos problemas básicos em grafos que pode ser resolvido por um algoritmo guloso e que também contempla características de um algoritmo por programação dinâmica, que será visto na Seção 3.5.

Problema dos Caminhos Mínimos (PCM): Dados grafo orientado $G = (V, E)$, onde cada arco $e \in E$ tem custo não-negativo c_e , um vértice $s \in V$, encontrar para cada $t \in V$, um caminho $P_{s,t}$ de s a t tal que $\sum_{e \in P_{s,t}} c_e$ é mínimo.

A estratégia utilizada pelo algoritmo a ser visto é muito próxima da utilizada no Algoritmo de Prim, para construção de uma árvore geradora de custo mínimo. No Algoritmo de Prim, a árvore começa com apenas o vértice raiz e a cada iteração a árvore é expandida anexando um novo vértice externo v . A estratégia gulosa usada pelo Algoritmo de Prim é iterativamente escolher um vértice que está mais próximo da árvore corrente.

No que segue, apresenta-se o Algoritmo de Dijkstra, que usa a mesma idéia do Algoritmo de Prim, mas em vez de anexar um vértice mais próximo da árvore, anexamos um que esteja mais próximo do vértice de origem s , uma vez que o problema busca por caminhos mínimos com origem em s . Com isto, o vetor $\text{dist}[v]$ mantém a melhor distância encontrada do vértice s até v . Inicialmente todos os vértices começam com valor $\text{dist}[v] = \infty$, exceto pelo vértice de origem que começa com $\text{dist}[s] = 0$. Estes valores iniciais garantem que o primeiro vértice a ser removido de Q é justamente o vértice de origem. Enquanto v está em Q , $\text{dist}[v]$ é atualizado sempre que se encontra um caminho melhor. Quando um vértice v é removido de Q , o caminho mínimo de s a v é definido e este não mais se altera.

Cada iteração do Algoritmo de Dijkstra também começa com uma árvore T , um vetor dist indexado nos vértices e uma fila de prioridade Q que prioriza vértices com menor dist e contém os vértices cuja busca de um caminho mínimo não foi finalizada. Neste caso, $\text{dist}[v]$ começa a iteração contendo a menor distância encontrada de um caminho de s para v , para todo $v \notin T$. Com isso, o algoritmo escolhe um vértice $v \notin T$ com $\text{dist}[v]$ mínimo. Feita a inclusão deste vértice, as estruturas de dados são atualizadas.

Em vez de representar os caminhos encontrados de s aos vértices de T , utiliza-se o vetor $\text{pred}[\cdot]$ para representar o percurso inverso, de um vértice de T a s . Se $v \in T$, o vértice $\text{pred}[v]$ é o vértice que precede v no caminho de s a v . Assim, o cami-

nho $(v, \text{pred}[v], \text{pred}[\text{pred}[v]], \dots, s)$ representa o caminho encontrado de s a v na ordem inversa. A árvore representada pelo vetor pred, é denominada *árvore de caminhos mínimos*, que como será visto, representará ao final do algoritmo, os caminhos mínimos em ordem invertida encontrados de um vértice s até outro vértice de G .

Uma descrição deste processo é apresentada no Algoritmo de Dijkstra. Note que a única diferença com o Algoritmo de Prim ocorre no bloco da linha 11, que atualiza dist e pred.

Algoritmo: Dijkstra(G, s) grafo ponderado $G = (V, E, c)$ e vértice $s \in V$.

Saída : Árvore de caminhos mínimos com origem em s .

```

1 para  $v \in V$  faca
2    $\text{dist}[v] \leftarrow \infty$ 
3    $\text{pred}[v] \leftarrow \text{nil}$ 
4    $\text{dist}[s] \leftarrow 0$ 
5    $Q \leftarrow V$ 
6    $S \leftarrow \emptyset$                                  $\triangleright$  usado apenas na prova
7 enquanto  $Q \neq \emptyset$  faca
8    $u \leftarrow \text{Extrai-Min}(Q)$ 
9    $S \leftarrow S \cup \{u\}$                            $\triangleright$  usado apenas na prova
10  para cada  $v \in \text{Adj}(u) \cap Q$  faca
11    se  $\text{dist}[u] + c(u, v) < \text{dist}[v]$  então
12       $\text{pred}[v] \leftarrow u$ 
13       $\text{dist}[v] \leftarrow \text{dist}[u] + c_{u,v}$ 
14 devolva ( $\text{dist}, \text{pred}$ )

```

O algoritmo explora a subestrutura ótima deste problema, quando considera que se um caminho de s a t é mínimo, qualquer subcaminho também é mínimo para os correspondentes extremos. O seguinte lema, cuja prova deixamos para o leitor, é válido.

Lema 3.4.3 *Seja $G = (V, E)$ um grafo orientado onde cada arco $e \in E$ possui um custo não-negativo c_e . Se $P = (v_1, v_2, \dots, v_k)$ é um caminho mínimo de v_1 a v_k , então o caminho $P' = (v_i, \dots, v_j)$ obtido de P , é um caminho mínimo de v_i a v_j , para $1 \leq i \leq j \leq k$.*

Naturalmente este resultado também vale quando a origem dos caminhos é sempre o vértice s . O seguinte lema também é válido e pode ser provado por indução.

Lema 3.4.4 *Se $d[v] < \infty$, então a sequência $v, \text{pred}[v], \text{pred}[\text{pred}[v]], \dots$ termina no vértice de origem s e representa a sequência inversa de um caminho de s a v de custo $d[v]$.*

Prova. Por indução no tamanho de S .

Base: Pelas atribuições iniciais, o primeiro vértice inserido em S possui $d[s] = 0$ e todos os demais possuem $d[v] = \infty$. Assim, s é o primeiro vértice inserido em S e o resultado é válido.

Hipótese: Suponha que o resultado é válido quando $|S| = k$.

Passo: Seja v o $(k+1)$ -ésimo vértice inserido em S . Se $d[v] = \infty$, não há nenhum arco saindo de um vértice de S e portanto não há caminho de s para v . Assim, $d[v]$ está calculado corretamente. Caso contrário, existe arco (u, v) de maneira que u entrou em S antes de v e $\text{pred}[v] = u$ e $d[v] = d[u] + c(u, v)$. Como há caminho de s para u tem-se que $d[v]$ é a soma do custo de um caminho de s até u e do custo do arco (u, v) . Portanto $d[v]$ é o custo do caminho definido de s até u e de u para v pelo arco (u, v) . \square

Teorema 3.4.5 *Dado grafo $G = (V, E)$, onde cada arco $e \in E$ tem custo não-negativo c_e , e vértice $s \in V$, o Algoritmo de Dijkstra(G, s) produz uma árvore de caminhos mínimos com origem em s .*

Prova. A prova usa o seguinte invariante: No início do laço da linha 7 $d[v]$ é o custo do caminho mínimo de s até v , para todo $v \in S$ e $(v, \text{pred}[v], \text{pred}[\text{pred}[v]], \dots, s)$ é um caminho de custo $d[v]$.

No início $S = \emptyset$ e o invariante é trivialmente válido. Suponha, para efeitos de contradição, que há alguma entrada para o qual o Algoritmo de Dijkstra computa incorretamente o caminho mínimo de um vértice, quando este foi inserido em S . Considere a execução neste grafo e seja u o primeiro vértice a ser inserido em S com custo calculado incorretamente. Certamente $u \neq s$, uma vez que $d[s]$ começa com 0, tendo sido o primeiro vértice de S . Seja P um caminho mínimo de s para u . Note que P deve ter algum vértice que não pertence a $S \cup \{u\}$. Seja y o primeiro vértice no caminho de P tal que $y \notin S$ e $x \in S$ o vértice que precede y em P . Seja P_x (resp. P_y) o caminho obtido de P saindo de s e chegando em x (resp. y). Pela subestrutura ótima, P_x e P_y são caminhos mínimos para seus respectivos vértices. Como u foi escolhido para ser removido antes de y , temos que $d[u] \leq d[y]$. Pela escolha de u , o caminho obtido pelo Algoritmo de Dijkstra para x é mínimo. No momento que x foi inserido em S , houve a atualização da estrutura percorrendo os vértices adjacentes a x .

Com isso $d[y]$ teve seu custo computado corretamente, uma vez que ao processar o arco (x, y) a partir de x , $d[y]$ receberia o valor $d[x] + c(x, y)$. Se existe algum arco de y para u com custo positivo, temos que $d[u] > d[y]$ e portanto y deveria ser removido de Q antes de u , o que contradiz a escolha de y . Assim, todos os arcos em P de y a u são nulos e portanto $d[y] \leq d[u]$. Portanto, $d[u] = d[y]$ e pelo lema anterior, o caminho dado por $(u, \text{pred}[u], \text{pred}[\text{pred}[u]], \dots, s)$ é um caminho válido de custo $d[u]$ e consequentemente também é de custo mínimo, contrariando a escolha de u . \square

A análise da complexidade de tempo do Algoritmo de Dijkstra depende da estrutura de dados usada para representar a fila de prioridades Q e é análoga a realizada para

o Algoritmo de Prim, descrito para o Problema da Árvore Geradora de Custo Mínimo. Neste caso, o Algoritmo de Dijkstra tem complexidade de tempo $O((|V| + |E|) \log |V|)$ usando heap e $O(|E| + |V| \log |V|)$ usando heap de Fibonacci.

3.4.3. Algoritmos Gulosos para os Problemas da Mochila Fracionária e Binária

O problema da Mochila Binária, como definido na Seção 3.2, admite apenas soluções onde um item está completamente incluso na mochila, ou o item não é incluído. A variante, conhecida como Problema da Mochila Fracionária, é análoga ao problema da mochila binária, porém, permite que se tome qualquer quantidade fracionária de um item.

Problema da Mochila Fracionária (PMF): Dados conjunto de itens $\{1, \dots, n\}$, cada item i com peso $p_i \geq 0$ e valor $v_i \geq 0$, ambos inteiros, e inteiro B , encontrar frações $x_i \in [0, 1]$, para $i = 1, \dots, n$, tal que $\sum_{i=1}^n p_i x_i \leq B$ e $\sum_{i=1}^n v_i x_i$ é máximo.

Note que a troca da restrição $x_i \in [0, 1]$ pela restrição $x_i \in \{0, 1\}$ nos dá exatamente o problema da mochila binária. Apesar de muito parecidos e ambos satisfazerem a propriedade de subestrutura ótima, os problemas da Mochila Binária e da Mochila Fracionária apresentam dificuldades computacionais bem distintas.

Considere o desenvolvimento de algoritmos gulosos para os dois problemas. Em ambos, a estratégia gúlota natural indica uma ordenação dos itens por seu peso relativo, que é o valor de um item por unidade de peso. Assim, considere que ambos já recebem itens em ordem que satisfaz $\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \dots \geq \frac{v_n}{p_n}$. Os algoritmos gulosos para estes dois problemas poderiam iterativamente colocar um item (ou a maior parte deste item, no caso fracionário) de maior valor relativo possível de ser incluído na solução. Caso um item (ou parte do item) não possa ser inserida, este é descartado. O algoritmo seguinte é um detalhamento desta idéia para o problema da mochila fracionária.

Algoritmo: Mochila-Fracionária-Guloso(B, p, v, n), onde $\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \dots \geq \frac{v_n}{p_n}$

Saída : Frações $x_i \in [0, 1]$ para $i = 1, \dots, n$.

- 1 $R \leftarrow B$
 - 2 **para** $i = 1$ até n **faz**
 - 3 $x_i \leftarrow \min\{R/p_i, 1\}$
 - 4 $R \leftarrow R - p_i x_i$
 - 5 **devolva** (x_1, \dots, x_n)
-

O algoritmo insere os itens na ordem dos que tem maior valor por unidade de peso primeiro. A cada iteração, o algoritmo procura colocar o máximo do item corrente e só então segue para o próximo item. Note que sempre que um item não couber em sua totalidade, a mochila é preenchida completamente e portanto todos os itens seguintes não serão colocados na solução. Assim, a propriedade da escolha gúlota é claramente

satisfeta. O próximo teorema, cuja prova fica como exercício, mostra que este algoritmo obtém uma solução ótima para o problema.

Teorema 3.4.6 *O Algoritmo Mochila-Fracionária-Guloso obtém uma solução ótima para o Problema da Mochila Fracionária.*

Agora, considere o Problema da Mochila Binária. O algoritmo guloso proposto anteriormente é descrito no Algoritmo Mochila-Guloso.

Algoritmo: Mochila-Guloso(B, p, v, n), onde $\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \dots \geq \frac{v_n}{p_n}$

Saída : Conjunto de itens S .

- 1 $R \leftarrow B$
 - 2 $S \leftarrow \emptyset$
 - 3 **para** $i = 1$ até n **faz**
 - 4 **se** $p_i \leq R$ **então**
 - 5 $S \leftarrow S \cup \{i\}$
 - 6 $R \leftarrow R - p_i$
 - 7 **devolva** S
-

Este algoritmo pode produzir soluções com valor arbitrariamente distantes do valor das soluções ótimas. O exemplo a seguir, mostra uma construção onde isto ocorre. Neste caso, a propriedade da escolha gulosa não é satisfeita.

Exemplo 3.4.1 Considere, por exemplo, uma entrada $I = (B, p, v, n)$ onde $B = 1$, $n = 2$, $v = (2\varepsilon, 1)$ e $p = (\varepsilon, 1)$, para $0 < \varepsilon < 1$. O peso relativo do primeiro item é $v_1/p_1 = 2\varepsilon/\varepsilon = 2$, enquanto o peso relativo do segundo é $v_2/p_2 = 1/1 = 1$. Ao colocar o primeiro item, não há espaço suficiente para o segundo, e temos uma solução gerada pelo algoritmo guloso de valor 2ε . A solução ótima é pegar apenas o segundo item, de valor 1. Portanto, a relação entre a solução ótima e a gerada pelo algoritmo é de $\frac{1}{2\varepsilon}$, que pode se tornar tão grande quanto se quiser, bastando para isso usar ε bem pequeno.

É interessante notar que apesar deste algoritmo guloso não fornecer nenhuma garantia sobre a solução gerada, é possível fazer uma pequena alteração para que garanta soluções com valor de pelo menos 50% do valor de uma solução ótima.

Exemplo 3.4.2 Seja t o primeiro item que o Algoritmo Mochila-Guloso não pode colocar no espaço remanescente da mochila e S^* uma solução ótima para uma entrada. Tome $S_1 = \{1, \dots, t-1\}$ e $S_2 = \{t\}$. É fácil ver que $\sum_{i=1}^t v_i \geq \sum_{i \in S^*} v_i$ é um limite superior para o valor de uma solução ótima. Como S_1 e S_2 são soluções, ao escolher a de maior valor, temos uma garantia de pelo menos $1/2$ do limite superior, e portanto de uma solução ótima.

Exemplo 3.4.3 Usando o mesmo tipo de argumento, que o usado no exemplo anterior, pode se mostrar que o algoritmo guloso produz soluções boas quando os itens não são grandes em relação à capacidade da mochila, em particular, quando $p_i \leq B/m$, para um parâmetro inteiro positivo m , que mede o quanto pequeno são os itens em relação ao recipiente. Quanto maior o valor de m , menor são os itens. Neste caso, o algoritmo guloso obtém soluções com valor pelo menos $\frac{m-1}{m}$ do valor da solução ótima.

3.5. Programação Dinâmica

O projeto de algoritmos por programação dinâmica também é um método indutivo. Com isso, poderiam ser resolvidos também pelo projeto por divisão e conquista (abordagem *top-down*). Porém, quando há subproblemas que se repetem na construção de um ou mais problemas, esta abordagem, sem maiores cuidados, pode levar a um algoritmo com complexidade de tempo muito maior que o necessário. Isto ocorre pela forma como é feita a resolução de cada um dos subproblemas, de maneira independente e sem conhecimento dos subproblemas já resolvidos. Um exemplo onde é possível perceber a velocidade como os subproblemas se multiplicam, é no cálculo da função de Fibonacci, quando esta é calculada por um algoritmo recursivo.

Exemplo 3.5.1 A função de Fibonacci $F(n)$ é definida nos inteiros positivos como $F(0) = 0$, $F(1) = 1$ e $F(n) = F(n - 1) + F(n - 2)$ se $n \geq 2$. Caso $F(n)$ seja implementado de maneira recursiva, o valor de $F(\cdot)$ será calculado repetidas vezes para várias entradas. A Figura 3.3 mostra os vários subproblemas envolvidos no cálculo de $F(4)$. Nota-se que $F(0)$ e $F(2)$ são resolvidos duas vezes cada um e $F(1)$ é resolvido por três vezes. A quantidade de operações realizadas para o cálculo do número

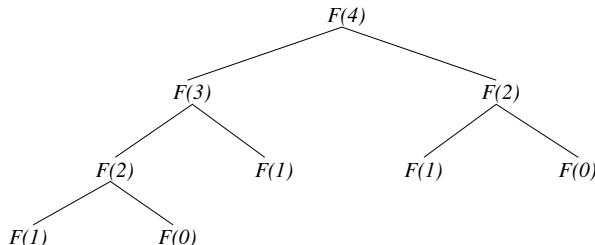


Figura 3.3. Resolução de $F(4)$ faz $F(0)$, $F(1)$ e $F(2)$ serem resolvidos mais de uma vez.

de Fibonacci pela abordagem top-down pode ser obtida resolvendo-se a recorrência de Fibonacci, que pelo Exemplo 3.3.4, mostra um crescimento exponencial para esta função. Mais precisamente, $F(n) = \Theta((1 + \sqrt{5})/2)^n$.

A estratégia usada no projeto por programação dinâmica é armazenar as soluções dos subproblemas já resolvidos em uma tabela. Caso um destes problemas repetidos apareça como subproblema de outro, obtém-se o resultado já armazenado, evitando

recalcular o mesmo subproblema. Os valores da função de Fibonacci, por exemplo, podem ser calculados do menor para o maior (abordagem *bottom-up*) e seus valores podem ser armazenados em um vetor. Como o cálculo de cada posição é feito em tempo constante, este processo obtém o valor de $F(n)$ em $\Theta(n)$ passos.

Em particular, o cálculo de um termo da sequência de Fibonacci depende apenas dos dois termos anteriores. Com isso, basta manter a cada iteração os dois maiores termos, pois após calcular o próximo termo, o menor termo não é mais usado, permitindo que o cálculo de $F(n)$ seja feito utilizando quantidade constante de espaço.

Os problemas para os quais a programação dinâmica se aplica têm, em geral, as seguintes propriedades.

Subestrutura ótima: Quando uma solução ótima de um problema deve conter soluções ótimas para seus subproblemas.

Subproblemas repetidos: presença de subproblemas que se repetem na resolução de diferentes problemas ou subproblemas.

Alguns passos importantes no desenvolvimento de um algoritmo por programação dinâmica são os seguintes [Cormen et al. 2009]: *(i)* Caracterização da estrutura de uma solução ótima. *(ii)* Definição recursiva do valor de uma solução ótima. *(iii)* Resolução dos problemas de maneira *bottom-up*. *(iv)* Construção da solução ótima.

As próximas subseções apresentam dois exemplos usando essa técnica, um para o Problema da Mochila Binária e outro para o Problema da Árvore Binária de Busca Ótima.

3.5.1. Algoritmo de Programação Dinâmica para o Problema da Mochila Binária

Esta seção apresenta um algoritmo por programação dinâmica para o Problema da Mochila Binária, definido na Seção 3.2.

Como pode ser visto, o problema da mochila binária contém a propriedade de subestrutura ótima. Considere uma solução ótima S^* para um problema da mochila binária de capacidade B . Ao remover um subconjunto $T \subseteq S^*$ da solução ótima e também o espaço no recipiente ocupado por estes itens, temos que $S^* \setminus T$ é uma solução ótima para o subproblema residual, formado pelos itens não pertencentes à T e com mochila de capacidade $B - \sum_{i \in T} p_i$.

Para simplificar o uso da subestrutura ótima, podemos considerar apenas um item e recair em problemas cujas soluções contém ou não tal item. Dada entrada $I = (B, p, v, n)$ e uma solução ótima S^* para I considere a pertinência do item n em S^* . Ocorre exatamente uma das situações: ou $n \in S^*$ ou $n \notin S^*$. Caso $n \in S^*$, recai-se no subproblema de completar o espaço restante da mochila da melhor maneira possível usando os demais $n - 1$ itens. Caso $n \notin S^*$, a mochila também deve ser preenchida na otimalidade com os demais $n - 1$ itens. Assim, de um problema com n itens recai-se em dois de $n - 1$ itens, sendo que um simples algoritmo recursivo teria complexidade

de tempo dado pela recorrência $T(n) = 2T(n - 1) + O(1)$, que resolvendo se obtém $T(n) \in \Theta(2^n)$. Tal complexidade de tempo não é muito melhor que o obtido por um algoritmo força-bruta. Para obter um algoritmo com complexidade de tempo melhor, pode-se explorar o fato que há subproblemas repetidos, que seriam resolvidos várias vezes em uma abordagem *top-down*, e do fato que na prática o tamanho da mochila não é muito grande.

O seguinte exemplo apresenta uma entrada para o problema da mochila binária que recai em subproblemas repetidos. Além disso, nota-se uma melhor definição dos subproblemas.

Exemplo 3.5.2 *Para entender a estrutura do Problema da Mochila Binária, considere a estratégia top-down aplicada a um problema com $B = 8$, e pesos $p = (1, 2, 1, 2, 1, 1, \dots)$. Os valores de cada item não foram colocados para não sobrecarregar a figura. Dado uma solução ótima, um item qualquer pode ou não pertencer a esta solução. Assim, um algoritmo usando a abordagem top-down pode considerar os dois casos, quando o item está ou não na solução, devolvendo uma que possuir o maior valor. A Figura 3.4 apresenta uma parte dos nós da árvore de ramificação de um possível algoritmo usando esta estratégia. Todos os subproblemas de um mesmo nível escolhem o mesmo item para fazer sua ramificação. Com isso, os subproblemas de um mesmo nível tem o mesmo conjunto de itens. Quando dois nós tiverem a mesma capacidade para a mochila residual, os nós tratam do mesmo problema. Disso, temos que o par de nós (a) e (d) representam o mesmo problema, assim como o par (b) e (e) e o par (c) e (f).*

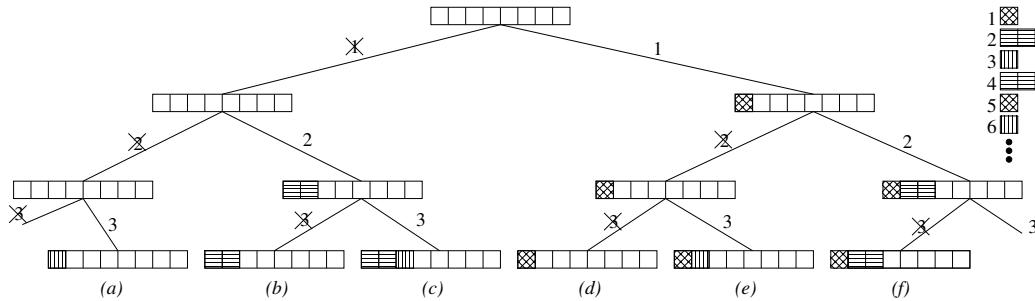


Figura 3.4. Problemas repetidos no problema da mochila: (a) e (d), (b) e (e) e (c) e (f).

Do exemplo anterior é possível notar que não basta ter o mesmo conjunto de itens para representar um mesmo subproblema. Para isto, é necessário que as capacidades das mochilas de cada subproblema também tenham o mesmo tamanho. Isto permite dizer melhor sobre o formato da entrada de cada subproblema.

Para a definição recursiva do problema, seja $I = (B, p, v, n)$ uma entrada para o problema da mochila binária. Considere o item n , que pode ou não estar em uma solução

ótima. Disso temos dois subproblemas, dados pelas entradas $I' = (B, p, v, n - 1)$, caso o item n não esteja na solução ótima sendo construída ou $I'' = (B - p_n, p, v, n - 1)$, quando o item n participa da solução. Os casos base se resumem a entradas onde não há itens ou a mochila não tem capacidade suficiente para novos itens.

Os subproblemas possíveis serão as entradas na forma (b, p, v, m) tal que $1 \leq b < B$ e $1 \leq m < n$. Portanto, o número de subproblemas que possuem pelo menos um item e tem capacidade não nula está limitado a nB . A definição recursiva do valor da solução ótima é a seguinte:

$$\vartheta(m, b) = \begin{cases} 0 & \text{se } m = 0 \text{ ou } b = 0, \\ \vartheta(m - 1, b) & \text{se } p_m > b, \\ \max\{\vartheta(m - 1, b), \vartheta(m - 1, b - p_m) + v_m\} & \text{se } p_m \leq b. \end{cases}$$

Com isso, a estratégia usada no algoritmo por programação dinâmica será resolver os subproblemas menores primeiro.

Itens	#Itens \ Mochila	0	1	2	...	$b - p_m$...	b	...	B
$\{1, \dots, n\}$	n	0			$\vartheta(n, B)$
\vdots	\vdots				
$\{1, \dots, m\}$	m	0			...			$\vartheta(m, b)$...	
$\{1, \dots, m - 1\}$	$m - 1$	0			...	$\vartheta(m - 1, b - p_m)$...	$\vartheta(m - 1, b)$...	
\vdots	\vdots				
$\{1, 2\}$	2	0			
$\{1\}$	1	0			
\emptyset	0	0	0	0	...	0	...	0	...	0

Tabela 3.2. Tabela de programação dinâmica para Mochila Binária preenchido bottom-up.

O Algoritmo Mochila-Tabela devolve a tabela dos valores dos subproblemas de (B, p, v, n) , mas não devolve o conjunto de itens que fazem parte da solução. Para encontrar os itens que compõem a solução ótima, partimos da posição (n, B) , onde se encontra o valor $\vartheta(n, B)$ da solução ótima e percorremos os itens na ordem reversa, decidindo a pertinência de cada item.

Na composição da solução de valor $\vartheta(n, B)$, o item n pode ou não ser utilizado. Se o item n não foi utilizado, o valor desta solução é igual a $\vartheta(n - 1, B)$ e a construção da solução continua a partir da célula $(n - 1, B)$. Caso contrário, a solução usa o item n e a busca segue a partir da célula $(n - 1, B - p_n)$. Isto é, podemos verificar se $\vartheta[n, B] = \vartheta[n - 1, B]$, e neste caso o item n não é usado. Caso contrário, o item n faz parte da solução ótima. Este processo é repetido até o primeiro item. O Algoritmo Mochila-PD descreve este processo.

Algoritmo: Mochila-Tabela(B, p, v, n).

Saída : Tabela dos subproblemas do problema da mochila binária.

- 1 **para** $b \leftarrow 0$ até B **faça** $\vartheta[0, b] \leftarrow 0$
- 2 **para** $m \leftarrow 0$ até n **faça** $\vartheta[m, 0] \leftarrow 0$
- 3 **para** $m \leftarrow 1$ até n **faça**
- 4 **para** $b \leftarrow 1$ até B **faça**
- 5 $\vartheta[m, b] \leftarrow \vartheta[m - 1, b]$
- 6 **se** $p_m \leq b$ e $v_m + \vartheta[m - 1, b - p_m] > \vartheta[m, b]$ **então**
- 7 $\vartheta[m, b] \leftarrow v_m + \vartheta[m - 1, b - p_m]$
- 8 **devolva** ϑ

Algoritmo: Mochila-PD(B, p, v, n).

Subrotina: Algoritmo Mochila-Tabela.

Saída : Solução ótima para o problema da mochila binária.

- 1 $\vartheta \leftarrow$ Mochila-Tabela(B, p, v, n)
- 2 $S \leftarrow \emptyset$
- 3 $b \leftarrow B$
- 4 **para** $m \leftarrow n$ decrescendo até 1 **faça**
- 5 **se** $\vartheta[m, b] \neq \vartheta[m - 1, b]$ **então**
- 6 $S \leftarrow S \cup \{m\}$
- 7 $b \leftarrow b - p_m$
- 8 **devolva** $S, \vartheta(n, B)$

O próximo teorema consolida o resultado do Algoritmo Mochila-PD, cuja prova de corretude ficará como exercício.

Teorema 3.5.1 *Dados entrada $I = (B, p, v, n)$ para o problema da mochila binária, o conjunto S , de valor $\vartheta(n, B)$ devolvido pelo Algoritmo Mochila-PD é uma solução ótima para a entrada I .*

A complexidade do Algoritmo Mochila-PD é claramente $O(nB)$. Trata-se de uma complexidade de tempo pseudo-polinomial, quando temos uma complexidade de tempo polinomial no tamanho da entrada e na magnitude do maior valor que ocorre na entrada. Quando B não é muito grande, é possível ter um algoritmo bastante eficaz na prática.

3.5.2. Algoritmo de Programação Dinâmica para o Problema da Árvore Binária de Busca Ótima

Uma árvore binária de busca, é uma estrutura de dados dinâmica que permite armazenar, fazer buscas, guardar a ordem e fazer operações básicas de manutenção de uma base de

dados, como remover, inserir e atualizar. Para tanto, seus elementos devem ter uma relação de ordem total. O conjunto dos elementos será denotado por \mathcal{D} .

Definida de maneira recursiva, uma *árvore binária* T é definida como: (i) Uma árvore vazia, representada por nil e não possui nenhum elemento nem subárvore; ou (ii) Um nó contendo um elemento $T.x$ e duas subárvores binárias, denotadas por subárvore esquerda de T , dada por $T.\text{esq}$, e subárvore direita de T , dada por $T.\text{dir}$. As duas subárvores são nós filhos da árvore T . A lista dos elementos representados por uma árvore binária T , denotado por $\mathcal{L}(T)$ é dado pela lista vazia, se $T = \text{nil}$ ou a lista $\mathcal{L}(T.\text{esq})\|(T.x)\|\mathcal{L}(T.\text{dir})$, caso T tenha pelo menos um elemento. Um nó também pode armazenar outras informações, porém, nos restrigiremos apenas às informações em \mathcal{D} . Uma *árvore binária de busca* é uma árvore binária definida sobre elementos de \mathcal{D} satisfazendo as seguintes propriedades: (a) \mathcal{D} é um conjunto com ordem total; (b) $T.x > e$, para todo $e \in \mathcal{L}(T.\text{esq})$; (c) $T.x < d$, para todo $d \in \mathcal{L}(T.\text{dir})$. Note que nesta definição não será permitido armazenar mais que um elemento de mesmo valor, caso necessário, estes deverão ser armazenados no mesmo nó.

Aplicações da árvore de busca ótima ocorrem na predição de palavras, busca em dicionários, verificadores de ortografia, tradução de textos, entre outros.

Considere, de maneira simplificada, o problema de se representar um dicionário de palavras. Neste caso, é importante que exista a possibilidade de mostrar o significado de uma palavra, mas também poder percorrer as palavras na ordem existente. Há estruturas de dados, como árvores 2-3 ou AVL que não só representam uma árvore de busca, mas também mantém a altura da árvore em ordem logarítmica no número de elementos, agilizando buscas. Porém, cada palavra em uma língua possui uma frequência em que é usada ou consultada. Com isso, é possível construir uma árvore binária de busca que leve esta frequência em consideração para produzir uma árvore que minimiza o número total de acessos aos seus nós, diminuindo com isso o tempo total das consultas.

Problema da Árvore Binária de Busca Ótima (PABO): Dados elementos $\mathcal{D} = (e_1 < e_2 < \dots < e_n)$, onde cada elemento $e \in \mathcal{D}$ é consultado $f(e)$ vezes, construa uma árvore binária de busca, tal que o total de nós consultados é mínimo.

Exemplo 3.5.3 Considere quatro chaves: $A < B < C < D$ e suas frequências $f(A) = 45$, $f(B) = 25$, $f(C) = 18$ e $f(D) = 12$. Há 19 árvores de busca binária válidas para estes quatro elementos. Na Figura 3.5 temos algumas destas, sendo que a primeira tem o menor custo de todos. É possível notar que há árvores com custos bem diversos.

Definicao 3.5.1 Se T é uma árvore binária de busca e v é um vértice de T , denotamos por $T(v)$ a subárvore enraizada em v contendo todos os vértices abaixo de v .

Exemplo 3.5.4 Para entender melhor a estrutura das soluções ótimas, considere uma árvore de busca contendo os elementos em $\{A, B, C, D\}$, onde $A < B < C < D$, e T é

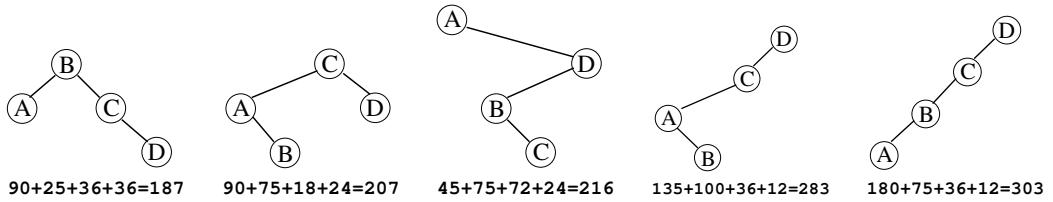


Figura 3.5. Nós acessados na primeira árvore: $2f(A)+f(B)+2f(C)+3f(D)=90+25+36+36=187$

uma árvore com três elementos, sendo o elemento $T.x = B$ como raiz e à sua esquerda, uma subárvore contendo apenas o elemento A e a sua direita uma subárvore contendo apenas o elemento D . Apesar de T representar uma árvore binária de busca, se no final todos os quatro elementos devem ser armazenados, então a árvore T não pode representar um subproblema válido. Dentre os elementos em $\mathcal{L}(T)$ falta o elemento C e em T há pelo menos um elemento maior que C e pelo menos um elemento menor que C . Com isso, se T fosse um subproblema, este deveria estar totalmente a esquerda de C ou totalmente a direita de C , porém isso não é possível, uma vez que T contém tanto elementos maiores como menores que C .

Portanto, as únicas subárvore de busca T , que representam subproblemas válidos, são as que definem $\mathcal{L}(T)$ como uma subsequência de elementos consecutivos da lista de elementos original ordenada.

Seja $\mathcal{D} = (e_1, e_2, \dots, e_n)$, onde $e_{k-1} < e_k$ para $k = 2, \dots, n$, a lista de elementos para o qual se deve construir uma árvore binária de busca. Do exemplo acima, conclui-se que uma árvore T só será um subproblema a ser considerado se existir i e j com $i \leq j$, tal que $\mathcal{L}(T) = (e_i, \dots, e_j)$.

Esta observação mostra que o número de subproblemas válidos distintos não é grande. De fato, é quadrático no número de termos total, uma vez que o número de subconjuntos de tamanho pelo menos dois em $\{e_1, e_2, \dots, e_n\}$ é $\binom{n}{2}$ e o número de subconjuntos de tamanho um é n . A estratégia será usar o projeto por programação dinâmica e armazenar os subproblemas de maneira que se possa computar um subproblema sabendo-se a solução dos subproblemas menores.

Denote por $T^*(e_i, \dots, e_j)$ uma árvore ótima dos elementos (e_i, \dots, e_j) . Se $i = j$, temos apenas um elemento e apenas uma árvore de busca binária pode representá-lo, e portanto é ótima. Caso contrário, sabe-se que um dos elementos em (e_i, \dots, e_j) deve ser a raiz de $T^*(e_i, \dots, e_j)$, e para saber qual pode ser, testamos todos os elementos e_k como raiz, para $k = i, \dots, j$.

Considere um dos testes quando e_k é raiz, onde $i \leq k \leq j$, e temos dois subproblemas filhos: $T^*(e_i, \dots, e_{k-1})$ e $T^*(e_{k+1}, \dots, e_j)$, como na Figura 3.6. Como cada um dos subproblemas filhos é menor, estes já foram computados e basta recuperar a solução ótima armazenada para cada um destes subproblemas. Porém, note que o valor

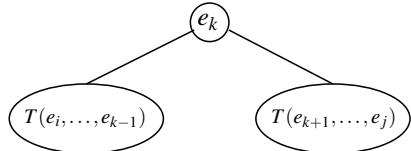


Figura 3.6. Divisão de um problema em subproblema.

armazenado nestes subproblemas não considera que cada consulta a um dos elementos nos subproblemas filhos, teve que consultar o nó e_k antes de chegar à árvore do subproblema. Portanto, deve-se acrescentar $\sum_{t=i}^{k-1} f(e_t)$ acessos ao nó e_k devido aos elementos do subproblema da esquerda e $\sum_{t=k+1}^j f(e_t)$ devido aos acessos dos elementos da direita. Além disso, há também os acessos ao nó e_k , igual a $f(e_k)$. Portanto, o número de acessos ao nó e_k é igual a $\sum_{t=i}^{k-1} f(e_t) + f(e_k) + \sum_{t=k+1}^j f(e_t)$. Agregando as somas, temos $\sum_{t=i}^j f(e_t)$ acessos ao nó e_k além dos acessos em cada subárvore/subproblema. Assim, se e_k é raiz de $T(e_i, \dots, e_j)$, temos um total de acessos para esta possibilidade igual ao obtido em $T^*(e_i, \dots, e_{k-1}) + T^*(e_{k+1}, \dots, e_j)$. Por fim, basta testar cada possibilidade de e_k ser raiz de uma subárvore com os elementos $\{e_i, \dots, e_j\}$ e escolher a melhor configuração.

Sabendo-se a estrutura de uma solução ótima, pode-se escrevê-la recursivamente. Como os subproblemas são dados por subsequências de elementos consecutivos, seja $A[i, j]$ a árvore de busca ótima do conjunto $\{e_i, \dots, e_j\}$.

$$A[i, j] = \begin{cases} 0 & \text{se } i > j, \\ f(e_i) & \text{se } i = j, \\ \min \left\{ A[i, k-1] + A[k+1, j] + \sum_{t=i}^j f(e_t) : k \in \{i, \dots, j\} \right\} & \text{se } i < j. \end{cases}$$

Para preencher a tabela $A[i, j]$ deve se tomar o cuidado para preencher na ordem dos menores para os maiores. Para isso, pode-se resolver os subproblemas de tamanho 1 (base), em seguida os subproblemas de tamanho 2, seguindo assim até o maior subproblema contendo n elementos.

Teorema 3.5.2 *O Algoritmo Árvore-Busca encontra o valor da árvore de busca ótima em tempo $O(n^3)$.*

A prova de corretude deste algoritmo bem como a construção da solução ficam como exercício.

3.6. Programação Linear e Inteira

Muitos problemas de otimização podem ser formulados matematicamente de modo que o objetivo se torne maximizar ou minimizar uma função linear definida sobre um conjunto de variáveis, as quais devem satisfazer um sistema de equações e inequações lineares. Neste caso, o sistema linear modela as restrições do problema que, sob esta forma,

Algoritmo: Árvore-Busca(e_1, \dots, e_n, f) onde $e_1 < e_2 < \dots < e_n$

Saída : Peso da melhor árvore de busca.

```
1 para  $i \leftarrow 1$  até  $n$  faça  $A[i, i] \leftarrow f(e_i)$ 
2 para  $t \leftarrow 2$  até  $n$  faça
3   para  $i \leftarrow 1$  até  $n - t + 1$  faça
4      $j \leftarrow i + t - 1$ 
5      $A[i, j] \leftarrow \min \{A[i, k - 1] + A[k + 1, j] : k \in \{i + 1, \dots, j - 1\};$ 
6            $A[i + 1, j]; A[i, j - 1]\} + \sum_{t=i}^j f(e_t)$ 
7 devolva  $A[1, n]$ 
```

define um problema de *programação linear*, no qual a função ser otimizada recebe o nome de *função objetivo*. Caso, adicionalmente, seja exigido que algumas ou todas as variáveis assumam apenas valores inteiros, tem-se um problema de *programação linear inteira*. A Programação Linear (PL) e a Programação Linear Inteira (PLI) constituem-se em poderosas ferramentas para a resolução de problemas de Otimização Combinatória. O intuito desta seção não é fazer uma apresentação exaustiva destes temas, mas sim destacar alguns aspectos relevantes destas técnicas para a Otimização Combinatória. Em particular, será enfatizada a modelagem de problemas combinatórios usando a PL e a PLI. Inicialmente, exemplifica-se o uso de PL na modelagem de um problema de otimização.

Exemplo 3.6.1 Suponha que uma companhia de geração de energia elétrica está planejando instalar uma usina termoelétrica em uma localidade. A maior dificuldade da empresa está em atender às exigências impostas pelas leis de proteção ambiental. Uma delas refere-se aos poluentes emitidos na atmosfera. O carvão necessário para aquecer as caldeiras deverá ser fornecido por três minas. As propriedades dos diferentes tipos de carvão produzidos em cada uma das minas estão indicadas na tabela abaixo. Os valores mostrados são relativos à queima de uma tonelada de carvão.

Mina	Enxofre (em ppm)	Poeira de Carvão (em Kg)	Vapor produzido (em Kg)
1	1100	1.7	24000
2	3500	3.2	36000
3	1300	2.4	28000

Os 3 tipos de carvão podem ser misturados em qualquer proporção. As emissões de poluentes e de vapor de uma mistura qualquer são proporcionais aos valores indicados na tabela. As exigências ambientais requerem que: (i) para cada tonelada de carvão queimada a quantidade de enxofre não deve ser superior a 2.500 ppm, e (ii) para cada tonelada de carvão queimada a quantidade de poeira de carvão não deve ser superior

a 2.8 kg. A companhia quer encontrar a proporção ótima da mistura do carvão de modo a maximizar a quantidade de vapor, portanto de energia, que é produzida com a queima de uma tonelada da mistura.

Para modelar o problema acima como um PL, pode-se definir as variáveis x_i , $i = 1, 2, 3$, como sendo a proporção do carvão da mina i que será usado na mistura. Deste modo, o valor a ser maximizado pode ser descrito pela função linear $z = 24000x_1 + 36000x_2 + 28000x_3$, que corresponde a energia produzida pela queima de uma tonelada da mistura de carvão nas proporções x_1 , x_2 e x_3 . De forma análoga, a limitação na quantidade de enxofre produzida pela queima de uma tonelada da mistura é descrita pela inequação $1100x_1 + 3500x_2 + 1300x_3 \leq 2500$, enquanto que a limitação da quantidade de poeira de carvão é dada por $1.7x_1 + 3.2x_2 + 2.4x_3 \leq 2.8$. Restam ainda duas restrições que precisam ser consideradas. A primeira delas diz respeito à proporção da mistura, a qual se supõe aqui não conter impurezas. Com isto, a seguinte igualdade precisa ser satisfeita: $x_1 + x_2 + x_3 = 1.0$. Para completar, todas as proporções devem ser não-negativas. Assim, o modelo final resultante é:

$$\begin{aligned} \max z &= 24000x_1 + 36000x_2 + 28000x_3 \\ \text{sujeito a } &1100x_1 + 3500x_2 + 1300x_3 \leq 2500, \\ &1.7x_1 + 3.2x_2 + 2.4x_3 \leq 2.8, \\ &x_1 + x_2 + x_3 = 1.0, \\ &x_1, x_2, x_3 \geq 0. \end{aligned}$$

De um modo geral, um problema de PL assume a forma abaixo:

$$\begin{aligned} \max z &= c_1x_1 + c_2x_2 + \dots + c_nx_n \\ \text{sujeito a } &a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \bowtie b_1, \\ &a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \bowtie b_2, \\ &\dots \\ &a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \bowtie b_m, \\ &x \in \mathbb{R}_+^n, \end{aligned} \tag{3}$$

onde o símbolo “ \bowtie ” pode representar uma das seguintes relações: “ $=$ ”, “ \leq ” ou “ \geq ”. Um modo mais compacto de representar um PL é a forma matricial dada por

$$\max\{cx : Ax = b, x \in \mathbb{R}_+^n\},$$

onde supõe-se que todas as restrições são de igualdade e as dimensões da matriz e dos vetores são: $A : m \times n$, $x : n \times 1$, $b : m \times 1$ e $c : 1 \times n$. Vale notar que, a rigor, qualquer problema de PL pode ser escrito de modo que todas restrições estejam sob a forma de igualdades ou sob a forma de desigualdades “ \leq ” (ou “ \geq ”, tanto faz). Isso é importante

pois alguns resultados para PL são demonstrados supondo que todas as relações são de um mesmo tipo, o que, pelo que acaba de ser dito, não os torna mais restritivos.

Para que um problema admita uma modelagem por PL, algumas hipóteses importantes devem ser satisfeitas. São elas: *(i)* proporcionalidade: a contribuição de uma variável na função objetivo e em uma restrição é multiplicada por k se o valor da variável também for multiplicado por k . *(ii)* aditividade: as contribuições individuais das variáveis se somam na função objetivo e nas restrições e são independentes, e *(iii)* divisibilidade: as variáveis podem ser divididas em qualquer fração.

Existem algoritmos que permitem resolver problemas de PL. O primeiro a surgir, e possivelmente o mais famoso e largamente empregado, é o método SIMPLEX desenvolvido por Dantzig em 1947 (cf., [Bazaraa et al. 2009]). Do ponto de vista da Teoria da Computação, o SIMPLEX apresenta um desvantagem por não ter complexidade polinomial, embora seu desempenho na prática seja altamente satisfatório. Passaram-se décadas desde o nascimento da PL até que aparecessem os primeiros algoritmos polinomiais, começando pelo *método dos elipsóides* [Khachiyan 1979] e com os *algoritmo de pontos interiores* [Karmarkar 1984]. Na prática, o SIMPLEX, mesmo tendo uma complexidade exponencial, ainda é competitivo, muitas vezes superando, algoritmos de pontos interiores. Existem bons resolvedores comerciais e de domínio público para resolver programas lineares, o que torna atrativo e viável o uso da técnica para a resolução de problemas de Otimização Combinatória.

A descrição dos algoritmos para PL foge ao escopo do presente texto. Contudo, é instrutivo que se tenha uma ideia sobre a geometria do problema, o que pode ser ilustrado no espaço bidimensional, ou seja, quando existem apenas duas variáveis x_1 e x_2 . Na Figura 3.7, o polígono hachurado corresponde às soluções do programa linear cujas restrições são $x_1 \geq 0$, $x_2 \geq 0$, $2x_1 + 6x_2 \leq 15$ e $2x_1 - 2x_2 \leq 3$. A função objetivo é dada por $\max z = 2x_1 + 3x_2$, cujo vetor gradiente está representado pela seta no alto à direita da figura. Como se sabe, à medida que se desloca na direção do gradiente, o valor da função z irá aumentar. Isso pode ser visto pelas linhas tracejadas que representam as retas da forma $2x_1 + 3x_2 = z$ para diferentes valores de z (indicados na figura). Ao se deslocar a reta na direção apontada pelo vetor gradiente, vê-se que o valor de z aumenta. Em um dado momento neste deslocamento, a reta atinge o ponto extremo do polígono cujo par de coordenadas é $(3, \frac{3}{2})$. A partir daí, se a reta continuar a se mover, ela não mais interceptará uma solução do programa linear. Conclui-se, portanto, que o ponto extremo em questão é uma solução ótima do programa linear e que o valor ótimo é $z = 10\frac{1}{2}$. Conforme ilustrado por este exemplo, o conjunto de vetores que satisfazem às restrições de um PL, chamadas de *soluções viáveis*, formam um polígono ou, mais geralmente, um poliedro (para dimensões maiores que 2). O conjunto de todas estas soluções (o poliedro) é chamado de *região de viabilidade* do PL. Além disso, como sugerido pelo exemplo, quando o valor ótimo for finito, existe pelo menos uma solução ótima que é um ponto extremo (vértice) do poliedro. Esta observação é a base do desenvolvimento

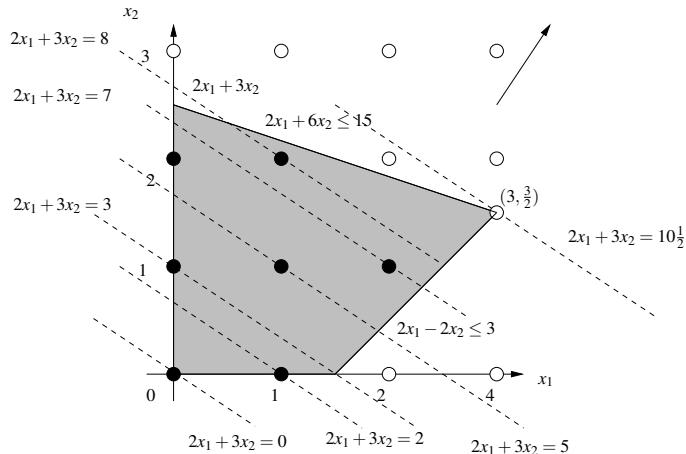


Figura 3.7. Representação Gráfica de um PL em duas variáveis.

do método SIMPLEX, o qual percorre sucessivamente vértices vizinhos da região de viabilidade, verificando se eles atendem a um critério de otimalidade.

Como visto acima, um PL pode ser resolvido em tempo polinomial. Contudo, quando se requer que uma ou mais variáveis tomem valores inteiros, a situação torna-se bem distinta. Isto porque, em sua forma geral, PLI está em NP -difícil, o que significa que, a menos que $\text{P} = \text{NP}$, não será possível encontrar um algoritmo polinomial para a PLI. Portanto, para resolver um problema de PLI na prática, recorre-se a algoritmos enumerativos, como será discutido na Seção 3.8. Vale notar que a PLI fere uma das hipóteses da PL que é a *divisibilidade* das variáveis. Para ilustrar este fato, dá-se a seguir um exemplo de um problema que é formulado por um PLI.

Exemplo 3.6.2 A administração de um hospital está montando a escala das enfermeiras que ficarão de plantão em um feriado prolongado. Estas enfermeiras serão responsáveis por procedimentos a serem realizados em pacientes que requerem cuidados especiais. O regime diário de trabalho das enfermeiras durante o feriado é tal que elas trabalham em turnos. Um período de trabalho é composto de 4 horas e um turno é composto de dois períodos de trabalho separados por uma pausa também de 4 horas, conforme exigências trabalhistas. Terminado o seu turno, uma enfermeira entra em um período de descanso de 12 horas seguidas. No total, são seis turnos cujos horários de trabalho ao longo do dia são mostrados na tabela a seguir. Nem todos os pacientes especiais passam por um procedimento em todos os períodos. Contudo, em um período qualquer, todo procedimento deve ser acompanhado o tempo todo por uma enfermeira. O número de procedimentos a serem feitos em cada período também é dado na tabela.

Turno	Período do dia (faixa horária)					
	00–04	04–08	08–12	12–16	16–20	20–24
1	✓		✓			
2		✓		✓		
3			✓		✓	
4				✓		✓
5	✓				✓	
6		✓				✓
Procedimentos a realizar	6	7	15	9	13	10

O objetivo da administração é encontrar o número mínimo de enfermeiras que devem ficar de plantão no feriado de modo a garantir que todos os procedimentos sejam realizados.

Para modelar este problema como um PLI, usamos as variáveis x_i para denotar o número de enfermeiras que ficarão no turno i . Com isto, o número de enfermeiras que serão chamadas para o plantão é dado por $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$. Já a quantidade de enfermeiras no período das meia-noite às 4 da manhã será $x_1 + x_5$ pois, para estar trabalhando neste período, uma enfermeira tem que estar sob o regime do turno 2 ou do turno 6. Raciocinando de forma análoga para os demais períodos do dia, chega-se ao seguinte modelo PLI:

$$\begin{array}{llllllllll} \min z = & x_1 & + & x_2 & + & x_3 & x_4 & + & x_5 & + & x_6 \\ \text{sujeito a} & x_1 & & & & & & + & x_5 & + & x_6 \\ & & x_2 & & & & & & & + & x_6 \\ & & & x_1 & + & x_3 & & & & & \geq 6, \\ & & & & x_2 & & + & x_4 & & & \geq 7, \\ & & & & & x_3 & + & & x_5 & & \geq 15, \\ & & & & & & x_4 & + & & x_6 & \geq 9, \\ & & & & & & & x_5 & + & & \geq 13, \\ & & x_1, & x_2, & x_3, & x_4, & x_5, & x_6 & \geq & 10, \\ & & x_1, & x_2, & x_3, & x_4, & x_5, & x_6 & \geq & 0, \\ & & & & & & & & & \text{inteiros.} \end{array}$$

A última condição é a chamada *restrição de integralidade* das variáveis do problema. Claramente a restrição de integralidade destrói a hipótese de divisibilidade da PL mas é imperativa neste caso, uma vez que a quantidade de enfermeiras em um turno não pode ser medido por um valor contendo uma parte fracionária não nula.

De um modo geral, para um PLI com n variáveis representadas pelo vetor $x \in \mathbb{Z}_+^n$, o PL obtido ao se substituir a restrição de integralidade pela restrição $x \in \mathbb{R}_+^n$ é denominado de *relaxação linear* do PLI. Tipicamente, ao resolver a relaxação linear, chega-se a uma solução ótima onde existem variáveis com valor fracionário (não inteiro). O valor ótimo da relaxação será, para um problema de maximização (minimização), um limite superior (inferior) ou *dual* para o PLI. A partir daí, percebe-se que se, por acaso, a solução ótima da relaxação contiver apenas variáveis de valor inteiro,

esta solução será ótima para o PLI. Feitas estas considerações, pode-se perguntar em que condições um PLI pode ser resolvido a partir da sua relaxação linear. Essa questão torna-se ainda mais relevante quando se analisa a complexidade computacional do problema que se está tratando. Essencialmente, se o PLI que modela um determinado problema de otimização combinatória pode ser resolvido como um PL, isto implica que o problema admite um algoritmo polinomial e, portanto, pertence à classe \mathbb{P} . No restante desta seção mostra-se como esta ideia pode ser aplicada à resolução de alguns problemas combinatórios conhecidos. Inicialmente, faz-se necessária a introdução de alguns conceitos adicionais.

Dada uma matriz $A : m \times n$, contendo apenas elementos 0, +1 ou -1, A é dita ser *totalmente unimodular* (TU) se toda submatriz quadrada de A tiver determinante 0, +1 ou -1. Abaixo são mostrados alguns exemplos que ilustram esta definição:

- Matrizes TU: $\begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$
- Matrizes não TU: $\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

A importância das matrizes TU para a PLI deriva do resultado a seguir.

Proposição 3.6.1 *Seja $S = \{x \in \mathbb{R}_+^n : Ax = b\}$ onde A é uma matriz totalmente unimodular e b é um vetor com todas as componentes inteiras. Então os pontos extremos de S são dados por vetores com todas coordenadas inteiras. Consequentemente, se $z = \max\{cx : x \in S\}$ tem um ótimo finito para um vetor $c \in \mathbb{R}^n$, este PL tem uma solução ótima que é inteira.*

Como mencionado, se o valor ótimo de um PL é finito, existe um ponto extremo da sua região de viabilidade que é uma solução ótima. Os algoritmos de PL são capazes de encontrar este ponto extremo ótimo. Agora, perceba que se o PL está associado à relaxação linear de um PLI particular, então, ao resolvê-lo, na verdade, se está resolvendo o PLI. Porém, isto significa que o problema de PLI em questão pode ser solucionado em tempo polinomial !

Do ponto de vista prático não faz muito sentido executar um algoritmo para identificar se a matriz A correspondente às restrições de um PLI genérico é TU ou não. Se ela for TU, o PLI será computado em tempo polinomial através de sua relaxação, porém isso poderia ser feito imediatamente, sem a necessidade de responder explicitamente se A é TU ou não. Mas, se A não for TU, geralmente algum algoritmo de complexidade exponencial terá que ser utilizado para achar uma solução ótima do PLI. Como será

visto na Seção 3.8, de qualquer modo estes algoritmos já precisam resolver a relaxação linear, independente da matriz A ser TU ou não. Contudo, do ponto de vista teórico, o conhecimento de que a matriz é TU é muito relevante. Além de prover de imediato uma forma prática de resolver o problema através dos algoritmos de PL, o que raramente será a forma mais eficiente de resolução, esta informação é uma prova de que o problema sendo investigado está na classe \mathbb{P} . Antes de prosseguir com exemplos em que esta situação ocorre, faz-se algumas considerações.

São conhecidas condições necessárias e suficientes para uma dada matriz ser TU. Contudo, neste texto apenas as condições necessárias expressas no resultado a seguir serão utilizadas para exemplificar situações em que um PLI pode ser resolvido por um algoritmo de PL. A importância deste resultado ficará evidente mais adiante.

Proposição 3.6.2 *Seja A uma matriz com todos elementos em $\{0, -1, +1\}$. Suponha que A contém no máximo dois elementos não-nulos por coluna e que existam dois subconjuntos disjuntos M_1 e M_2 do conjunto M dos índices das linhas de A tal que, $M_1 \cup M_2 = M$ e, para toda coluna j contendo dois coeficientes não-nulos, tem-se que $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$. Então, A é TU.*

Além destas condições necessárias as propriedades a seguir também podem ser facilmente verificadas.

Proposição 3.6.3 *As seguintes afirmações são equivalentes: (i) A é TU; (ii) A^T (matriz transposta de A) é TU; (iii) $\begin{pmatrix} A \\ I \end{pmatrix}$, ou seja, a matriz A acrescida das linhas da matriz identidade I , é TU.*

Pela Proposição 3.6.2, pode-se verificar que a matriz A de incidência vértice-arco de um grafo direcionado $G = (V, E)$ é TU. Suponha que $|V| = n$ e $|E| = m$ de modo que A tenha dimensão $n \times m$. Dado um arco $e = (i, j)$, os elementos da e -ésima coluna de A são todos nulos, exceto aqueles correspondentes às linhas i e j cujos valores são $+1$ e -1 , respectivamente. Portanto, a matriz só tem elementos 0 , $+1$ e -1 , uma condição necessária para ser TU, e cada coluna tem exatamente dois elementos não nulos como requerido na hipótese da Proposição 3.6.2. Agora, sendo $M = \{1, \dots, n\}$ os índices das linhas, pode-se fazer $M_1 = M$ e $M_2 = \emptyset$. Pela definição da matriz e dos conjuntos M_1 e M_2 , para toda coluna e correspondente a um arco em E , tem-se que $\sum_{i \in M_1} a_{ie} = \sum_{i \in M_2} a_{ie} = 0$. Logo $\sum_{i \in M_1} a_{ie} - \sum_{i \in M_2} a_{ie} = 0$ para toda coluna e , e, sendo assim, A é TU.

Sabendo que a matriz de incidência vértice-arco de um grafo direcionado é TU, pode-se mostrar que o *Problema de Fluxo de Custo Mínimo* (PFCM) em uma rede com dados de entrada inteiros está na classe \mathbb{P} . Inicialmente, define-se o PFCM formalmente.

Problema do Fluxo de Custo Mínimo (PFCM):

Entrada: grafo direcionado (rede) $G = (V, E)$; demandas b_i para todo $i \in V$, satisfa-

zendo $\sum_{i \in V} b_i = 0$, se b_i for positivo (negativo) esta demanda será a quantidade de fluxo injetada (retirada) da rede no vértice i , enquanto que b_i nulo significa que i é apenas um vértice de passagem para o fluxo; custos c_{ij} por unidade de fluxo que passa no arco (i, j) , para todo arco em E ; capacidades h_{ij} que limitam o fluxo máximo que pode passar no arco (i, j) , para todo arco em E ;

Solução: deve dizer a quantidade de fluxo que passa em cada arco de modo a escoar todo fluxo injetado na rede nos vértices i com $b_i > 0$ (fontes) para que este chegue nos vértices onde haja retirada de fluxo (sorvedouros), ou seja, aqueles com $b_i < 0$. A quantidade de fluxo em cada arco é não-negativa e não pode ser superior a sua capacidade.

Objetivo: minimizar o custo total do fluxo que passa pelos arcos da rede.

Denotando por x_{ik} o fluxo que passa pelo arco (i, k) e por $V^+(i)$ ($V^-(i)$) os vértices j em $V \setminus \{i\}$ para os quais o arco (i, j) ((j, i)) está em E , o problema pode ser modelado como um PL da seguinte forma:

$$\begin{array}{ll} \min & z = \sum_{(i,j) \in E} c_{ij} x_{ij} \\ \text{sujeito a} & \sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = b_i, \quad \forall i \in V, \end{array} \quad (4)$$

$$x_{ij} \leq h_{ij}, \quad \forall (i, j) \in E, \quad (5)$$

$$x_{ij} \geq 0, \quad \forall (i, j) \in E. \quad (6)$$

As equações (4) são chamadas de *restrições de conservação de fluxo* nos vértices e impõem que as quantidades totais de fluxo que saem e que chegam de qualquer vértice da rede tem que ser iguais. Note que o fluxo injetado (retirado) em um vértice i com $b_i > 0$ ($b_i < 0$) entra na (sai da) rede por este vértice. As restrições (5) apenas limitam o fluxo máximo que passa em cada arco da rede, enquanto as restrições (6) dizem que este mesmo fluxo não pode ser negativo.

Observe que a submatriz formada apenas pelos coeficientes das restrições (4) corresponde exatamente à matriz de incidência vértice-arco do grafo (direcionado) G . Como visto anteriormente, esta matriz é TU. Ao anexar a esta matriz as linhas correspondentes aos coeficientes da submatriz formada pelas restrições (5), o que se faz é estender a matriz com as linhas de uma matriz identidade de ordem $|E|$. Pelo item (ii) da Proposição 3.6.3, tem-se que a matriz de restrições do PFCM é TU. Assim, supondo que todos os b'_i 's e h_{ij} 's sejam inteiros, pela Proposição 3.6.1, sabemos que a solução do problema linear que modela o PFCM é inteira. De acordo com a discussão anterior, fica provado que o PFCM com demandas e capacidades inteiras pode ser resolvido em tempo polinomial.

Este último resultado tem sua relevância ampliada quando se percebe que importantes problemas combinatórios são casos particulares do PFCM. Este é o caso do *problema do caminho mais curto* entre um vértice s e um vértice t de um grafo direcionado, ou ainda, o *problema do fluxo máximo* entre os vértices s e t neste mesmo grafo.

A seguir discute-se como estes problemas são modelados pelo PFCM. Antes porém, ambos são definidos formalmente.

Dado um grafo direcionado $G = (V, E)$, dois vértices distintos s e t de V e uma função $l : E \rightarrow \mathbb{R}_+$ que associa comprimentos aos arcos de E , deseja-se encontrar um caminho de s para t em G tal que a soma dos comprimentos dos arcos neste caminho seja mínima. Recorde que um caminho de um vértice s a um outro vértice t no grafo G é uma sequência de vértices $(v_0, v_1, v_2, \dots, v_p)$ tal que $v_0 = s$, $v_p = t$ e $(v_{i-1}, v_i) \in E$ para todo $i = 1, \dots, p$. O problema do caminho mais curto (PCMC) descrito acima pode ser resolvido por algoritmos combinatórios de complexidade polinomial em $n = |V|$ e $m = |E|$, como o Algoritmo de Dijkstra, apresentado na Seção 3.4.2. Contudo, para evidenciar como a PL e as matrizes TU podem ajudar a estabelecer a complexidade de um problema, pelo menos no que diz respeito à sua pertinência à classe \mathbb{P} , suponha por um momento que a complexidade do PCMC é desconhecida.

Para formular o PCMC como um PFCM, considere inicialmente que o comprimento de um arco seja equivalente ao custo unitário de passar uma unidade de fluxo por ele. Agora, suponha que se deseja escoar uma unidade de fluxo de s para t . Na terminologia do PFCM, isto corresponde a dizer que $b_s = 1 = -b_t$ que e $b_i = 0$ para todo $i \in V \setminus \{s, t\}$. O fluxo que deixa s deve percorrer um caminho em G até chegar em t . O custo desta operação será a soma dos custos dos fluxos nos arcos do caminho, já que a quantidade de fluxo que passa é unitária. Contudo, por construção, esta soma é exatamente o comprimento do caminho. Logo, ao minimizar o custo do escoamento do fluxo, na verdade, também se está resolvendo o problema de encontrar o caminho de comprimento mínimo de G que liga s a t . O modelo de PL para o PCMC seria:

$$\begin{array}{ll} \min & z = \sum_{(i,j) \in E} c_{ij}x_{ij} \\ \text{sujeito a} & \sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = 0, \quad \forall i \in V \setminus \{s, t\}, \end{array} \quad (7)$$

$$\sum_{k \in V^+(s)} x_{ik} - \sum_{k \in V^-(s)} x_{ki} = 1, \quad (8)$$

$$\sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = -1, \quad (9)$$

$$x_{ij} \leq 1, \quad \forall (i, j) \in E, \quad (10)$$

$$x_{ij} \geq 0, \quad \forall (i, j) \in E. \quad (11)$$

Conforme argumentado no modelo geral do PFCM, a matriz dos coeficientes das restrições da formulação acima é TU. Como o vetor correspondente ao termo independente das restrições (o “lado direito” das mesmas) possui todas componentes inteiras, pela Proposição (3.6.1), este modelo admite uma solução ótima que é inteira, no caso, um vetor binário. Ou seja, o caminho mais curto será composto pelos arcos (i, j) para os

quais a variável x_{ij} tem valor 1 na solução ótima. A consequência deste fato é que o PCMC está em \mathbb{P} . Isto porque ele tem um número de restrições e variáveis que é polinomial em n e m e todos os coeficientes da matriz de restrições e do vetor de termos independentes estão em $\{0, +1, -1\}$. Logo, o modelo pode ser resolvido por um algoritmo de pontos interiores, ou pelo método dos elipsóides em tempo polinomial no tamanho da entrada do PCMC.

No problema do fluxo máximo tem-se o seguinte cenário. Na entrada é dado um grafo direcionado $G = (V, E)$ e dois vértices distintos s e t em V . Por hipótese, supõe-se que o arco (t, s) não esteja em E (veja em [Ahuja et al. 1993] formas simples de alterar a rede no caso em que este arco está E). Além disso, para cada arco (i, j) de E , é dada uma capacidade positiva h_{ij} . O *problema do fluxo máximo* (PFM) pede que seja encontrada a quantidade máxima de fluxo que ao ser injetada para dentro da rede através do vértice s consegue atingir o vértice t . O fluxo injetado em s percorrerá os arcos da rede, sempre respeitando suas capacidades máximas. Como no caso anterior, este problema admite algoritmos de complexidade polinomial no tamanho da entrada mas será suposto, apenas para fins didáticos, que não é sabido que PFM está em \mathbb{P} .

Ao tentar modelar este problema como um PFCM, surge uma dificuldade, já que as demandas de fluxo nos vértices não são dados de entrada. A rigor, o que se busca é um valor máximo para b_s ou, equivalentemente para $-b_t$, de modo que, se $b_i = 0$ para todos os demais vértices, o PFCM ainda admita uma solução viável. Para contornar esta situação, faz-se uma pequena alteração no grafo G , acrescentando-se a E o arco (t, s) com capacidade de fluxo ilimitada. Denote-se por $G' = (V, E' = E \cup \{(t, s)\})$ o grafo resultante desta operação. Além disso, fixe $b_i = 0$ para todo vértice i de V , incluindo s e t . Finalmente, atribua custo nulo para a circulação de uma unidade de fluxo a todos os arcos, exceto ao novo arco (t, s) ao qual se atribui o custo 1. Note que ao fazer isso, se na rede original G for escoado um fluxo de f unidades de s para t , na rede G' , esta mesma quantidade de fluxo deverá retornar a s através do arco (t, s) , já que a rede está isolada do mundo externo pelo fato das demandas em todos os vértices serem nulas. Seguindo esta interpretação, o PFM pode ser resolvido se encontrarmos o maior valor possível para f , respeitadas as capacidades dos arcos originais de G . Logo, definido-se x_{ij} como sendo a quantidade de fluxo que passa no arco (i, j) para todo $(i, j) \in E'$, o PFM pode ser modelado pelo seguinte problema de PL:

$$\begin{array}{ll} \max & z = x_{ts} \\ \text{sujeito a} & \sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = 0, \quad \forall i \in V, \end{array} \quad (12)$$

$$x_{ij} \leq h_{ij}, \quad \forall (i, j) \in E = E' \setminus \{(t, s)\}, \quad (13)$$

$$x_{ij} \geq 0, \quad \forall (i, j) \in E'. \quad (14)$$

Caso as capacidades h_{ij} sejam dadas por valores inteiros, assim como ocorreu com o PCMC, tem-se um PL cuja matriz de restrições é TU. Novamente conclui-se que o

problema admite um algoritmo polinomial no tamanho de entrada via a PL. Isso seria, portanto, uma prova de que o PFM (com capacidades inteiras nos arcos) está em \mathbb{P} .

3.7. Algoritmos *Backtracking*

Esta seção apresenta o projeto de algoritmos por backtracking, que é bastante aplicado no desenvolvimento de algoritmos exatos para problemas de otimização combinatória. A técnica também consiste na busca de uma solução ótima por enumeração, construindo as candidatas a solução de maneira incremental, adicionando ou descartando elementos a cada chamada recursiva ou iteração. Diferente de um algoritmo força-bruta, a técnica busca limitar a quantidade de ramificações a explorar, podando ramos que não levam a soluções melhores. Para isto, é necessário investigar a estrutura combinatória do problema procurando propriedades que permitam identificar ramos não promissores. Diminuindo a quantidade de ramos percorridos de maneira efetiva, é possível encontrar uma solução ótima em menos tempo.

Nas próximas seções, a técnica é aplicada a dois problemas: ao problema da Mochila Binária e ao problema do Conjunto Independente Máximo.

3.7.1. Algoritmo *Backtracking* para o Problema da Mochila Binária

O algoritmo proposto para o PMB é recursivo e a cada chamada considera-se um item e duas situações, uma quando o item é adicionado à solução corrente e outra quando o item não pertence à solução.

Inicialmente, considere um conjunto de itens cuja soma dos pesos ultrapassa a capacidade da mochila. Naturalmente, trata-se de um conjunto inviável para o problema da Mochila Binária e ao percorrer por um subconjunto que o contém, este será igualmente inviável e também deve ser descartado. Como a técnica constrói soluções de maneira incremental, a poda dos conjuntos inviáveis é feita de maneira direta. Se a cada chamada, um item é colocado apenas se este couber no espaço remanescente, então, claramente o algoritmo fará sua enumeração passando apenas por soluções factíveis.

Outra estratégia usada para otimizar a busca, é evitar fazer a busca por direções que, apesar de possuirem soluções factíveis, estas tem valores iguais ou piores que a melhor solução corrente. Para explorar essa estratégia, é importante ter soluções boas o quanto antes, e saber estimar, durante o processo de ramificação, o quanto se pode melhorar a partir daquele ponto. Esta estimativa representa um limitante para o valor da melhor solução possível que completa a solução corrente. Caso esta estimativa indique haver apenas soluções iguais ou piores que a melhor solução obtida até o momento, evita-se fazer a busca por tal ramo. Caso não se tenha encontrado soluções até o momento, ou a atual candidata a solução não tenha boa qualidade, haverá menos chances de um ramo ser podado. Segue disto a importância de se ter boas soluções o quanto antes, podendo inclusive começar a enumeração já com o valor de uma solução candidata obtida por métodos heurísticos. Por exemplo, pode-se usar o algoritmo guloso, visto para o PMB, como heurística para a obtenção de uma primeira solução viável. De fato,

o algoritmo apresentado usa uma estratégia mais sofisticada e aplica a própria busca gulosa também como parte do processo de enumeração. Mais precisamente, a cada chamada recursiva, o algoritmo escolhe um item, dentre os que não foram considerados, com o maior valor relativo. Caso este item possa ser inserido no espaço remanescente, o algoritmo busca primeiro pelo ramo que incorpora o item na solução corrente e posteriormente, busca pelo ramo que o descarta.

Para percorrer os itens na mesma ordem que a utilizada pelas chamadas recursivas, basta ordená-los previamente por seu valor relativo, obtendo $\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \dots \geq \frac{v_n}{p_n}$. Desta forma, a ramificação é guidada por uma direção gulosa, na esperança de encontrar soluções melhores mais rapidamente.

Para que se possa podar um ramo não-promissor, é necessário obter algum limitante para a qualidade das soluções que serão obtidas a partir deste ramo. No caso do problema da Mochila Binária, é possível obter um limitante para uma solução ótima através de uma solução para o problema da correspondente Mochila Fracionária. Assim, suponha que no início de uma chamada recursiva consideramos o m -ésimo item, para colocar ou não na solução corrente. Seja x^* a melhor solução obtida até o momento e x a solução atual. O valor e peso total dos itens representados em x^* (resp. x) é dado por $V^* = v \cdot x^*$ e $P^* = p \cdot x^*$ (resp. $Vx = v \cdot x$ e $Px = p \cdot x$). O espaço remanescente neste ponto é dado por um valor $B' = B - p \cdot x$ e um limitante superior para este espaço é obtido pela aplicação do algoritmo que resolve o problema da Mochila Fracionária com capacidade B' e itens em $\{m, \dots, n\}$. Se a solução deste problema da Mochila Fracionária é dada por V_f , o limitante para qualquer solução que complementa a atual solução é no máximo $v \cdot x + V_f$. Caso este valor seja no máximo o valor da melhor solução corrente $v \cdot x^*$, podemos descartar os ramos que seguem do nó corrente. Este processo é descrito no Algoritmo Mochila-BT.

A primeira chamada da rotina recebe, além dos dados de entrada, dois parâmetros: um vetor n -dimensional x^* com todas as posições iguais a zero e um vetor sem elementos, x . O vetor x^* mantém a melhor solução obtida durante a execução da rotina e o vetor x representa a solução que está sendo construída. A cada nova chamada recursiva, o vetor é incrementado com mais uma posição. Observa-se que, no passo 3 pode-se utilizar uma estimativa mais fraca V'_f , definida como $V'_f = (B - p \cdot x) \frac{v_m}{p_m}$. Desta forma, o cálculo da estimativa é feito em tempo constante, e apesar de ser uma estimativa mais fraca, pode ser melhor em alguns casos, dado que o tempo computacional para seu cálculo é menor.

Apesar do Algoritmo Mochila-BT ter complexidade de tempo $O(2^n)$, o algoritmo possui um bom desempenho prático. Não é possível dizer que este algoritmo é mais rápido ou mais lento que o Algoritmo Mochila-PD, apresentado na Seção 3.5.1, que utiliza programação dinâmica, uma vez que cada um pode ser mais rápido para diferentes entradas. Por fim, pode-se implementar o Algoritmo Mochila-BT de maneira iterativa. Recomenda-se que o leitor implemente a versão iterativa deste algoritmo.

Algoritmo: Mochila-BT(B, p, v, n, x^*, x, m) onde $\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \dots \geq \frac{v_n}{p_n}$

Saída : Atualiza x^* se encontrar solução melhor que completa x

- 1 **se** $m \leq n$ **então**
 - 2 **se** $v \cdot x > v \cdot x^*$ **então** $x^* \leftarrow x$
 - 3 **seja** V_f o valor da solução de
Mochila-Fracionária-AG($B - p \cdot x, v[m, \dots, n], p[m, \dots, n], n - m + 1$)
 - 4 **se** $v \cdot x + V_f > v \cdot x^*$ **então**
 - 5 **se** $p_m \leq B - p \cdot x$ **então** Mochila-BT($B, p, v, n, x^*, x \parallel (1), m + 1$)
 - 6 Mochila-BT($B, p, v, n, x^*, x \parallel (0), m + 1$)
-

3.7.2. Algoritmo Backtracking para o Problema do Conjunto Independente

Nesta seção, investiga-se o problema do Conjunto Independente Máximo, para o qual, é apresentado uma sequência de algoritmos, onde cada um pode ser visto como uma sofisticação do anterior. São apresentados alguns algoritmos por backtracking para resolver o problema do conjunto independente máximo, onde a cada nova versão, temos um algoritmo que explora melhor a estrutura combinatória do problema. Esta seção foi baseada em [Kloks 2012, Erickson 2015].

Definicao 3.7.1 Dado grafo $G = (V, E)$ não-orientado, um subconjunto $S \subseteq V$ é um conjunto independente de G se quaisquer dois vértices distintos de S não estão ligados por uma aresta de G .

Problema do Conjunto Independente (PCIM): Dado grafo não-orientado G , encontrar um conjunto independente de G de cardinalidade máxima.

Um algoritmo por força bruta para o PCIM pode ser feito de maneira análoga ao feito para o algoritmo força bruta para o PMB. Neste caso, usa-se um vetor binário $x \in \{0, 1\}^{|V|}$, onde $x_v = 1$ se v está na solução e $x_v = 0$ caso contrário. Ao testar todos os vértices do grafo, verifica-se se o conjunto representado é um conjunto independente maior que o encontrado até o momento, e neste caso o atualiza. É possível verificar se o vetor x representa um conjunto independente em tempo linear no tamanho do grafo, isto é, em $O(n + m)$, que ficará como exercício para o leitor. Com isso, a complexidade de tempo deste algoritmo é $O(2^n(n + m))$, isto é, $O^*(2^n)$.

O objetivo será fazer um algoritmo com complexidade de tempo melhor que do força bruta. Considere uma versão de algoritmo backtracking para o PCIM, enumerando a pertinência de cada vértice na solução. Em cada iteração, pode se escolher um vértice qualquer $v \in V$ e encontrar dois conjuntos independentes máximos, um condicionado a não ter o vértice v e outro condicionado a tê-lo. Naturalmente um deles é um conjunto independente máximo de G . No primeiro caso, a busca recai no grafo $G - v$. No segundo

caso, v deve pertencer à solução, e consequentemente os vértices adjacentes a V não podem pertencer. Esta idéia é representada no seguinte algoritmo.

Algoritmo: ConjInd1(G)

Saída : Um conjunto independente de $G = (V, E)$ de cardinalidade máxima.

- 1 **se** $|V| \leq 1$ **então devolva** V
 - 2 **senão**
 - 3 seja $v \in V$ um vértice qualquer.
 - 4 $S_0 \leftarrow \text{ConjInd1}(G - v)$.
 - 5 $S_1 \leftarrow \text{ConjInd1}(G - \text{Adj}(v) - v) \cup \{v\}$.
 - 6 **devolva** $S \in \{S_0, S_1\}$, com $|S|$ máximo.
-

Como não necessariamente há vértices adjacentes a v , ambas as chamadas para a rotina ConjInd1, nos passos 4 e 5, podem ocorrer para um grafo com $n - 1$ vértices, onde $n = |V|$. Com isso, a complexidade computacional do Algoritmo ConjInd1 é dada por $T_1(n) = 2T_1(n - 1) + p_1(n)$, onde $p_1(n)$ é a função de complexidade computacional que delimita o tempo gasto pela rotina ConjInd1, sem contar o tempo das chamadas recursivas. Considerando estruturas adequadas para representar G é possível que $p_1(n)$ seja polinomial em n . Portanto, pelo Teorema 3.3.2, temos que $T_1(n)$ é $O^*(2^n)$, o que não leva a melhorias no termo exponencial, em relação ao algoritmo força bruta.

Para desenvolver um algoritmo com complexidade de tempo melhor, deve-se explorar melhor a estrutura combinatória do problema. De fato, para cada uma das chamadas recursivas das linhas 4 e 5, considerou-se que os grafos das chamadas recursivas diminuiu apenas um vértice. Porém, na chamada da linha 5, isto só ocorre quando v é escolhido ser vértice isolado, sem arestas incidentes. Naturalmente, todos os vértices isolados pertencem a um conjunto independente de cardinalidade máxima. Com isso, sempre que há tais vértices, estes podem ser incorporados ao conjunto sendo construído.

Considere um algoritmo obtido do Algoritmo ConjInd1, denominado aqui por ConjInd2, que, logo antes da escolha do vértice v , incorpora todos os vértices isolados na solução. Com isto, v será adjacente a pelo menos um vértice e portanto, a chamada da linha 5 será feita para um grafo com no máximo $n - 2$ vértices. Assim, a recorrência que representa o comportamento assintótico deste algoritmo é dada por

$$T'_2(n) \leq \max\{T'_2(n - 1); T'_2(n - 1) + T'_2(n - 2)\} + p_2(n),$$

onde $p_2(n)$ é a função de complexidade computacional do tempo gasto pela rotina ConjInd2, sem contar o tempo das chamadas recursivas. Naturalmente, a recorrência $T'_2(n)$ pode ser limitada pela recorrência $T_2(n) = T_2(n - 1) + T_2(n - 2)\} + p_2(n)$, que aplicando o Teorema 3.3.2, tem-se que $T_2(n)$ é $O^*(1.618\dots^n)$, uma melhora significativa em relação ao primeiro algoritmo.

Buscando melhorar o Algoritmo ConjInd2, considere o caso crítico deste algoritmo. Na análise acima, este ocorre quando o vértice v possui exatamente um vértice adjacente. Seja u o vértice adjacente a v e S um conjunto independente de cardinalidade máxima de G . Note que necessariamente u ou v deve pertencer a S , mas não ambos. Se u pertence a S , podemos obter um novo conjunto independente de mesma cardinalidade removendo u e adicionando v . Assim, concluímos que neste caso, sempre há um conjunto independente máximo contendo v . Logo, podemos incorporá-lo na solução e fazer a chamada recursiva para o grafo sem os vértices u e v . A descrição deste algoritmo é dada a seguir.

Algoritmo: ConjInd3(G)

Saída : Um conjunto independente de $G = (V, E)$ de cardinalidade máxima.

- 1 **se** $|V| \leq 1$ **então devolva** V
- 2 **senão**
- 3 **se** existe $v \in V$: $\text{grau}(v) \leq 1$ **então**
- 4 **devolva** ConjInd3($G - \text{Adj}(v) - v$) $\cup \{v\}$.
- 5 **senão**
- 6 Seja $v \in V$ um vértice qualquer.
- 7 $S_0 \leftarrow \text{ConjInd3}(G - v)$.
- 8 $S_1 \leftarrow \text{ConjInd3}(G - \text{Adj}(v) - v) \cup \{v\}$.
- 9 **devolva** $S \in \{S_0, S_1\}$, com $|S|$ máximo.

Analisando a complexidade de tempo do Algoritmo ConjInd3, considerando que o grafo usado na chamada da linha 8 tem no máximo $n - 2$ vértices, é possível limitar a complexidade de tempo deste algoritmo, de maneira análoga a feita para delimitar T'_2 por T_2 , pela recorrência

$$T_3(n) = T_3(n-1) + T_3(n-3) + p_3(n),$$

onde $p_3(n)$ é um polinômio em n . A resolução desta recorrência nos dá que $T_3(n)$ é $O^*(1.465\dots)^n$.

Neste ponto está claro que teremos uma complexidade de tempo melhor, se garantirmos que o vértice v , escolhido para ser considerado dentro ou fora do conjunto independente, tem grau tão grande quanto possível. O caso crítico do Algoritmo ConjInd3 ocorre quando o vértice v , escolhido no passo 6, tem grau 2. Agora, considere uma versão de algoritmo que escolhe um vértice v de grau máximo. Enquanto o vértice escolhido tiver grau pelo menos 3, a complexidade de tempo do algoritmo deve levar a uma função de complexidade de tempo melhor que a apresentada por T_3 . Mas, inevitavelmente podemos chegar a situação onde não há vértices de grau pelo menos 3. Por outro lado, tal grafo possui estrutura bastante restrita. O seguinte lema, cuja prova fi-

cará como exercício, mostra que de fato é possível encontrar um conjunto independente neste grafo de maneira eficiente.

Lema 3.7.1 *Se G é um grafo onde todos os vértices tem grau no máximo 2, então é possível obter um conjunto independente de cardinalidade máxima em tempo linear.*

Algoritmo: ConjInd4(G)

Saída : Um conjunto independente de $G = (V, E)$ de cardinalidade máxima.

- 1 **se** $|V| \leq 1$ **então devolva** V
- 2 **senão**
- 3 **se** $\text{grau}(v) \leq 2$ *para todo* $v \in V$ **então**
- 4 *Seja* S *um conjunto independente de cardinalidade máxima de* G .
- 5 **devolva** S .
- 6 **senão**
- 7 *Seja* $v \in V$ *um vértice de* G *de grau máximo.*
- 8 $S_0 \leftarrow \text{ConjInd4}(G - v)$.
- 9 $S_1 \leftarrow \text{ConjInd4}(G - \text{Adj}(v) - v) \cup \{v\}$.
- 10 **devolva** $S \in \{S_0, S_1\}$, com $|S|$ máximo.

De forma análoga, o tempo de execução deste algoritmo pode ser limitado pela seguinte recorrência

$$T_4(n) = T_4(n-1) + T_4(n-4) + p_4(n),$$

onde $p_4(n)$ é um polinômio em n . Resolvendo esta recorrência, temos que T_4 é $O^*(1.380\dots^n)$.

Alguns pontos de interesse das análises de complexidade de tempo dos algoritmos de tempo exponencial, descritos acima.

- Alterar o esforço computacional para dividir e/ou conquistar os subproblemas, mantendo este esforço em tempo polinomial, sem alterar o tamanho dos subproblemas, não mudará a base da complexidade de tempo exponencial.
- Porém, se o esforço investido diminuir o tamanho das entradas usadas nas chamadas recursivas, pode-se diminuir a base da complexidade de tempo exponencial.

Do ponto de vista teórico, estes pontos indicam que é melhor investir na redução do tamanho dos subproblemas, contanto que esta redução seja feita em tempo polinomial.

3.8. Algoritmos *Branch and Bound*

O *branch-and-bound* é uma técnica empregada na resolução de problemas difíceis de otimização combinatória, i.e., aqueles para os quais não se conhecem algoritmos polinomiais capazes de resolvê-los. Um algoritmo *branch-and-bound* enumera *implicitamente* todas as soluções do problema. Evidentemente, como o conjunto de soluções é, na maioria dos casos, exponencial no tamanho da entrada, esta enumeração deve ser *ímplicita* pois, do contrário, o algoritmo seria de pouca utilidade prática, equivalendo-se à resolução do problema pela *força bruta*. Os algoritmos *branch-and-bound* são exemplos de aplicação do paradigma de divisão-e-conquista uma vez que a prova de otimalidade da solução retornada por eles se baseia na partição sucessiva do conjunto de soluções. O termo *branch*, palavra inglesa que pode ser traduzida por *ramificação*, diz respeito a este processo de partição. Já o termo *bound* refere-se ao fato dos algoritmos *branch-and-bound* valerem-se do cálculo de limitantes duais e primais para o valor ótimo procurado. São estes limitantes que permitem que se possa fazer uma busca inteligente por uma solução ótima, evitando a enumeração exaustiva de todas as possíveis soluções. Para melhor entender o funcionamento de um algoritmo *branch-and-bound* é necessário introduzir algumas definições e terminologias.

O processo de particionamento do conjunto de soluções efetuado por um algoritmo *branch-and-bound* pode ser representado por uma *árvore de enumeração*, também denominada por *árvore de espaço de estados*. Nesta árvore, os nós folha representam as soluções enquanto os nós internos dizem respeito aos subconjuntos de soluções que são considerados à medida que o processo de particionamento sucessivo avança. Assim, um nó interno está associado ao subconjunto de soluções que correspondem às folhas da subárvore com raiz naquele nó. Consequentemente, de acordo com esta interpretação, o nó raiz representa o conjunto de todas as possíveis soluções do problema.

Agora, considere o problema de otimização combinatória sob a forma de maximização¹ dado por: $z = \max\{cx : x \in S\}$, onde S é o conjunto de todas as soluções viáveis do problema. Suponha uma partição $\{S_1, S_2, \dots, S_K\}$ de S de modo que $z^k = \max\{cx : x \in S_k\}$, para todo $k = 1, \dots, K$. Não é difícil ver que $z = \max_k z^k$, ou seja, o valor ótimo global é o máximo dos valores ótimos computados sobre os elementos da partição. Além disso, suponha que, para todo $k = 1, \dots, K$ sejam conhecidos um limitante dual (superior) \bar{z}^k e um limitante primal (inferior) \underline{z}^k para z^k . Pode-se verificar que os valores de \bar{z} e \underline{z} abaixo são limitantes superior e inferior de z , respectivamente,

$$\bar{z} = \max_k \{\bar{z}^k\} \quad \text{e} \quad \underline{z} = \max_k \{\underline{z}^k\}.$$

A observação anterior permite entender o princípio básico do algoritmo *branch-and-bound*. Partindo do nó raiz, o algoritmo faz uma exploração da árvore de enumeração. Supondo que ele compute limitantes inferiores e superiores a cada nó que explore,

¹A maior parte desta seção trata de problemas de maximização. Não é difícil adaptar os resultados para o caso de minimização.

pelo resultado acima ele terá constantemente à sua disposição um limitante dual e um primal para o valor ótimo z . Este último pode ser calculado, por exemplo, por meio de heurísticas. A rigor, se estes limitantes forem iguais², isto significa que a solução que fornece o limitante primal \underline{z}^k é ótima. Nesta situação, o algoritmo para, interrompendo-se a exploração da árvore de enumeração. Evidentemente que quanto menor a quantidade de nós explorados, menor será o tempo de execução do algoritmo e, portanto, maior será a sua eficácia. Daí a enorme importância da qualidade tanto dos limitantes duais quanto dos primais para viabilizar o uso de um algoritmo *branch-and-bound*. Esta qualidade é mensurada pelo desvio destes valores em relação ao ótimo.

Antes de prosseguir com mais detalhes relativos ao funcionamento do algoritmo, discute-se sobre as formas alternativas de efetuar o particionamento do conjunto de soluções. Dependendo como isto é feito, a árvore de enumeração poderá ter formas diferentes. Por exemplo, se o conjunto de soluções representadas por um nó interno for sempre particionado em dois subconjuntos, a árvore de enumeração será binária, já que este nó terá dois filhos. Contudo, diferentes estratégias de particionamento do conjunto de soluções de um nó interno podem ser adotadas, originando árvores onde estes nós têm um número de filhos que pode ser fixo (e diferente de 2) ou até mesmo variável. Esta ideia é ilustrada nos exemplos a seguir, onde considera-se que o nó raiz da árvore de enumeração encontra-se no nível zero.

Exemplo 3.8.1 Considere o problema da mochila binária definido sobre 3 itens em que a solução é representada por um vetor binário x de tamanho 3, em que $x_i = 1$ se e somente se o item $i \in \{1, 2, 3\}$ é levado na mochila. A árvore de enumeração é construída de modo que, no nível i , as decisões sobre os i primeiros itens já estão fixadas. Assim, estando em um nó interno no nível i da árvore de enumeração, pode-se fazer uma ramificação em que este nó dá origem a dois filhos, um deles à esquerda que será a raiz de uma subárvore onde todas soluções não irão conter o item i , ou seja, $x_i = 0$. Analogamente, a subárvore à direita corresponderá ao subespaço de soluções em que o item i será incluído na mochila e, portanto, $x_i = 1$. A árvore de enumeração seria aquela vista na Figura 3.8. Evidentemente, neste caso a árvore é binária.

Exemplo 3.8.2 Considere o problema do caixeiro viajante, definido na seção 3.2.1, sobre um grafo direcionado completo contendo 4 vértices³. Inicialmente, note que a descrição de um circuito hamiltoniano pode ser feita, sem ambiguidades, pela ordem em que os vértices são visitados ao começar o percurso no vértice 1. A partir daí, define-se a árvore de enumeração de modo que em um nó interno no nível i , considera-se que já tenham sido tomadas as decisões sobre os $i + 1$ primeiros vértices visitados no circuito. Com isto, o nó terá $n - i - 1$ ramificações possíveis, cada uma associada

²Será visto mais adiante que em algumas situações esta condição pode ser relaxada

³O termo “nó” será sempre usado nesta seção para referir-se à árvore de enumeração, enquanto “vértices” será reservado para grafos de entrada de problemas que estejam sendo tratados.

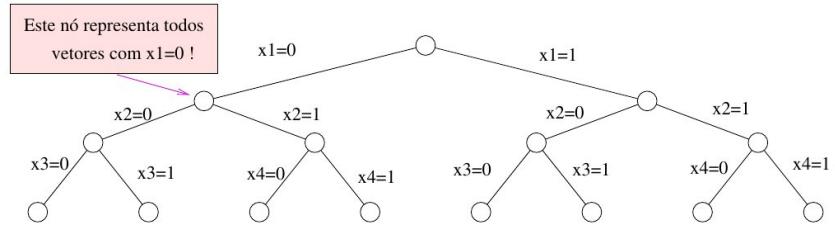


Figura 3.8. Árvore de enumeração completa para o problema binário da mochila com 3 itens.

a um possível vértice que irá suceder o último vértice do caminho parcial (no grafo) que sai do vértice 1 e corresponde às ramificações percorridas ao se sair da raiz da árvore de enumeração até chegar ao nó interno em questão. Para um grafo completo de 4 vértices, a árvore de enumeração seria aquela vista na Figura 3.9. Repare que nesta árvore, o grau de um nó no nível i é sempre igual a $n - i - 1$.

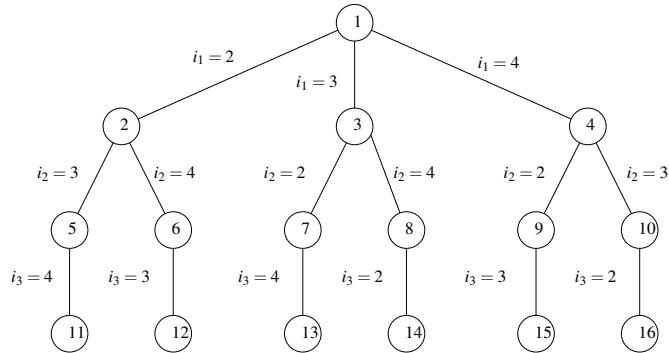


Figura 3.9. Árvore de enumeração completa para o problema do caixeiro viajante de um grafo direcionado completo com $n = 4$ vértices, onde $i_0 = i_4 = 1$.

Tendo visto exemplos de diferentes formas de particionamento do espaço de soluções e das árvores de enumeração que delas resultam, retorna-se agora à descrição do funcionamento do algoritmo *branch-and-bound*. Antes de prosseguir, deve-se ter em mente que a árvore de enumeração é uma abstração e que, obviamente, ela não é armazenada explícitamente na memória. No entanto, o fluxo do algoritmo *branch-and-bound* coincide com a porção da árvore que foi explorada que, para fins de eficácia, deve ser a menor possível. Dito isso, analisa-se agora como usar os limitantes primais e duais para *podar* ramos da árvore, evitando explorar subárvores que não contenham a solução ótima. Ou seja, ao *podar* um nó⁴ impede-se que o algoritmo faça alguma

⁴Também se usa o termo *amadurecer* um nó.

ramificação a partir dele. São três os tipos de poda, a saber: por *optimalidade*, *limitante* e por *inviabilidade*. Estas 3 situações são ilustradas nos exemplos a seguir. Nestes exemplos, considera-se que a existência de um limitante primal pressupõe o conhecimento de uma solução viável cujo valor corresponde àquele limitante. Além disso, supõe-se que o problema que se esteja resolvendo seja de maximização.

Exemplo 3.8.3 Considere a porção da árvore de enumeração mostrada na Figura 3.10, onde S representa o conjunto de soluções correspondente ao nó pai e S_1 e S_2 é uma partição de S . Os limitantes primais (inferiores) e duais (superiores) encontram-se ao lado dos respectivos nós.

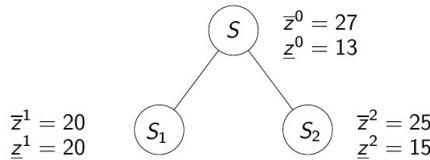


Figura 3.10. Exemplo de poda por optimalidade.

Como visto anteriormente, os limitantes superior e inferior para S são:

$$\bar{z} = \max\{\bar{z}^1, \bar{z}^2\} = \max\{20, 25\} = 25 \quad e \quad \underline{z} = \max\{\underline{z}^1, \underline{z}^2\} = \max\{20, 15\} = 20$$

Observe que em S_1 é conhecida uma solução de custo 20 ($= \underline{z}^1$) e ela é ótima para a subárvore com raiz neste nó, pois o custo de qualquer solução de S_1 é ≤ 20 ($= \underline{z}^1$). Logo o nó que representa S_1 deve ser podado.

Exemplo 3.8.4 Considere agora a porção da árvore de enumeração mostrada na Figura 3.11. Neste caso, a solução de maior custo em S_1 tem custo limitado a 20 ($= \bar{z}^1$)

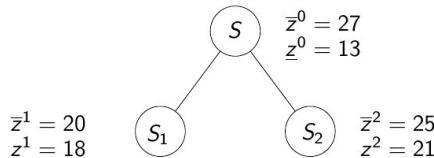


Figura 3.11. Exemplo de poda por limitante.

mas, em S_2 já é conhecida uma solução de custo 21 ($= \underline{z}^2$). Logo o nó que representa S_1 deve ser podado, pois a exploração desta subárvore só pode conduzir a soluções subótimas.

Exemplo 3.8.5 Para ilustrar a poda por inviabilidade, considera-se a instância do problema da mochila binária para 4 itens cuja restrição é dada por $8x_1 + 5x_2 + 3x_3 + 3x_4 \leq 12$. Considere agora a porção da árvore de enumeração mostrada na Figura 3.12, na qual supõe-se que o espaço de estados é particionado como no Exemplo 3.8.1. O nó S_4 representa o subconjunto de todas as soluções que contêm simultaneamente os itens 1 e 2. Mas este subconjunto é vazio já que a soma dos pesos destes itens ultrapassa a capacidade da mochila. Consequentemente este nó deve ser podado.

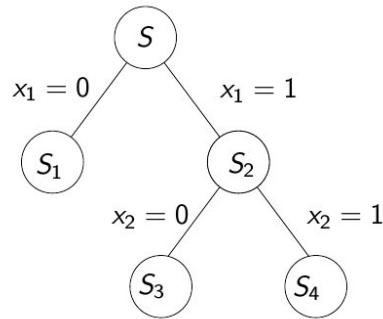


Figura 3.12. Exemplo de poda por inviabilidade.

Uma vez definidos os tipos de poda, os nós da porção da árvore de enumeração explorados pelo algoritmo *branch-and-bound* se dividem em três grupos. No primeiro deles encontram-se os nós que foram ramificados, como é o caso do nó identificado por S nos exemplos anteriores. No segundo grupo estão os nós podados. Finalmente, o último grupo é composto pelos chamados *nós ativos*, aqueles que não foram podados e, portanto, poderão ser ramificados.

A estratégia de exploração da árvore adotada pelo algoritmo depende fundamentalmente da escolha do próximo nó ativo a ser ramificado a cada iteração. Pode-se imaginar que os nós ativos correntes são armazenados em uma lista, inicialmente composta exclusivamente pelo nó raiz. Um pseudocódigo para um algoritmo *branch-and-bound* genérico é exibido na Figura 3.13. Nele a lista de nós ativos é representada pela variável *Ativos*. Vê-se, portanto, que a escolha do próximo nó a ramificar é feita na linha 4 do algoritmo. Comumente, na prática, tal escolha feita de acordo com a estratégia do *melhor limitante*, a qual consiste em optar pelo nó cujo limitante superior é o maior. O princípio que norteia esta estratégia é que, de acordo com os limitantes duais calculados até aquela iteração, a subárvore com raiz naquele nó é aquela que potencialmente apresenta uma solução com o maior valor. Não é difícil perceber que, ao adotar esta estratégia, nunca serão explorados nós com limitantes duais piores do que o valor ótimo. Claramente, ao se fazer isso pretende-se minimizar o número de nós explorados. Outras

critérios de escolha são propostos na literatura e que podem ser convenientes em situações específicas. É o caso, por exemplo, da adoção da busca em profundidade quando se resolve um problema onde há muita dificuldade na obtenção de uma solução primal.

1. B&B; (* considerando problema de **maximização** *)
2. Ativos $\leftarrow \{\text{nó raiz}\}$; melhor-solução $\leftarrow \{\}$; $\underline{z} \leftarrow -\infty$;
3. **Enquanto** (Ativos não está vazia) **faça**
 4. Escolher um nó k em Ativos para ramificar;
 5. Remover k de Ativos;
 6. Gerar os filhos de k : n_1, \dots, n_q computando \bar{z}_{n_i} e \underline{z}_{n_i} correspondentes;
(* definir \bar{z}_{n_i} e \underline{z}_{n_i} iguais a $-\infty$ para subproblemas inviáveis *)
 7. **Para** $j = 1$ até q **faça**
 8. **se** ($\bar{z}_{n_j} \leq \underline{z}$) **então** podar o nó n_j ; (* inclui os 3 casos *)
 9. **se não**
 10. **Se** (n_j representa uma única solução) **então** (* atualizar melhor limitante primal *)
 11. $\underline{z} \leftarrow \bar{z}_{n_j}$; melhor-solução $\leftarrow \{\text{solução de } n_j\}$;
 12. **se não** adicionar n_j à lista Ativos.

Figura 3.13. Algoritmo *branch-and-bound* genérico.

Pelo exposto acima, nota-se que alguns aspectos são relevantes para a implementação de um algoritmo *branch-and-bound*. No caso dos limitantes, tanto duais como primais, a questão é se devem ser usados limitantes fáceis de calcular, usualmente fracos, ou limitantes fortes mas computacionalmente caros. Também deve-se pensar nas formas de decomposição do espaço de soluções e em como isto impacta o armazenamento e a manutenção da porção da árvore de enumeração explorada pelo algoritmo. Ademais, como visto acima, também é preciso definir uma ordem de percurso da árvore. Independentemente disso, existe uma quantidade mínima de informações que precisam ser armazenadas para a correta operação do algoritmo. Isso incluiu os limitantes superiores de todos os nós ativos e o limitante inferior global \underline{z} .

Considera-se agora a aplicação do algoritmo *branch-and-bound* a 3 problemas.

3.8.1. Algoritmo *Branch-and-bound* para o Problema da Mochila Binária

Como mencionado, os limitantes duais são cruciais para o bom desempenho do algoritmo *branch-and-bound*. Tais limitantes podem ser gerados através de relaxações do problema que se quer resolver e para as quais são conhecidos algoritmos eficientes de resolução. Para o problema da mochila, uma relaxação usual é obtida ao se permitir que sejam levadas frações dos itens na mochila. Como visto na Seção 3.4 o problema da mochila fracionária pode ser resolvido em tempo polinomial usando um algoritmo guloso. Assim, um algoritmo *branch-and-bound* para a mochila binária poderia ser projetado em que a ramificação dos nós segue o esquema do Exemplo 3.8.1 e o cálculo do limi-

tante dual seria feito pelo algoritmo guloso. Considere então a instância do problema da mochila dada por:

$$\begin{aligned} \max \quad & 8x_1 + 16x_2 + 20x_3 + 12x_4 + 6x_5 + 10x_6 + 4x_7 \\ \text{s.t. } & 3x_1 + 7x_2 + 9x_3 + 6x_4 + 3x_5 + 5x_6 + 2x_7 \leq 17 \\ & x_i \in \{0, 1\}, i = 1, \dots, n. \end{aligned}$$

Note que os $n = 7$ itens já se encontram ordenados em ordem não crescente da relação custo/peso, facilitando rápida identificação da solução ótima das relaxações.

A Figura 3.14 mostra a parte explorada da árvore de espaço de estados, considerando o uso da estratégia de melhor limitante (em casos de empate foram privilegiados os nós com maior profundidade e, persistindo o empate, a ordem de geração do nó). Ao lado de cada nó exibe-se uma tripla da forma (W', C, \bar{z}_{n_i}) onde W' é a capacidade residual da mochila, C é o custo da solução parcial correspondente ao nó e \bar{z}_{n_i} é o valor do limitante obtido pelo algoritmo guloso naquele nó (já consideradas as variáveis fixadas pelas ramificações que levam até ele). Nesta figura, as ramificações de um nó interno no nível i significam que o item i é levado ($x[i] = 1$) na solução quando se tratar do filho esquerdo e, do contrário, que ele não é levado ($x[i] = 0$) no caso do filho direito. Os números no interior dos nós indicam a ordem em que eles foram explorados, começando em zero. A ordem de geração dos nós foi: 0, 1, 16, 2, 15, 3, 4, 8, 5, 6, 7, 9, 11, 10, 12, 13, 14. Os nós podados por inviabilidade são representados por pequenos círculos pretos. Assim, por exemplo, saindo da raiz e tomando-se sempre o ramo da esquerda, chega-se a um nó preto que corresponde a todas as soluções em que $x_1 = x_2 = x_3 = 1$. Contudo, a soma dos pesos destes três itens é 19, ou seja superior à capacidade da mochila que é 17. Daí a poda por inviabilidade. Os nós 7, 13 e 14 correspondem a soluções inteiras (poda por otimalidade), enquanto os nós 15 e 16 foram podados por limitante. Pecebe-se claramente a vantagem do algoritmo *branch-and-bound* sobre o uso da força bruta, tendo em vista que a árvore de enumeração completa neste caso possui $2^{7+1} - 1 = 255$ nós e só foram explorados 23 !

Note a proximidade do algoritmo por branch-and-bound com a versão backtracking. A principal diferença é na forma como os nós são explorados. No Algoritmo Mochila-BT, a ramificação é guiada por uma heurística gulosa, usando apenas a pilha de recursão para percorrer a árvore de maneira a privilegiar ramos mais promissores.

Na versão branch-and-bound há mais liberdade para se explorar os ramos. A versão apresentada dá prioridade para o nó com melhor limitante. Esta escolha favorece o estreitamento do *gap* entre o limitante superior e inferior, melhorando a garantia da qualidade da solução ao longo do tempo. Porém, nem sempre atinge soluções boas. Uma hibridização das duas formas pode buscar por ramos promissores enquanto não há uma solução primal boa e, após isso, concentrar-se em fechar o *gap* entre os limitantes.

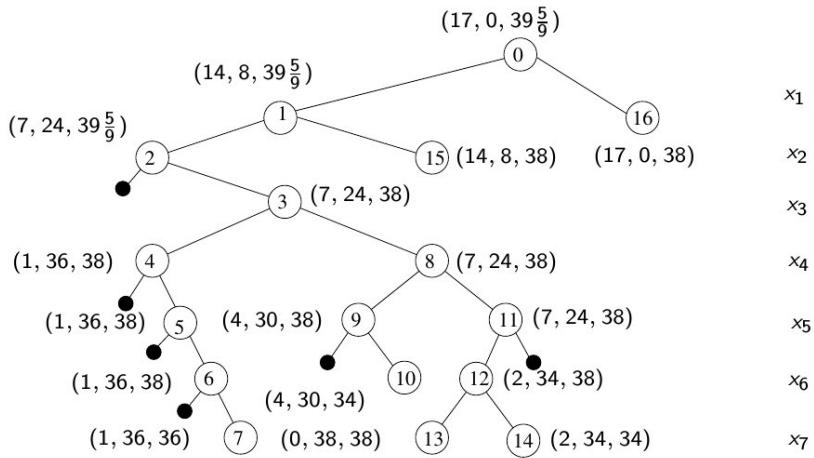


Figura 3.14. Aplicação do algoritmo *branch-and-bound* ao problema da mochila binária.

3.8.2. Algoritmo *Branch-and-bound* para um Problema de Escalonamento de Tarefas

Considere o seguinte problema de escalonamento de tarefas em máquinas. Na entrada são dados um conjunto de n tarefas J_1, \dots, J_n e duas máquinas M_1 e M_2 . Cada tarefa é composta de duas operações sendo que a primeira deve ser executada em M_1 e a segunda em M_2 , somente após encerrada a execução da primeira operação. O tempo de processamento da tarefa J_j na máquina M_i é dado por t_{ij} . O *tempo de término* de execução da i -ésima operação da tarefa J_j na máquina M_i é dado por f_{ij} . Cada máquina só pode executar uma operação por vez e uma vez iniciada uma operação em uma máquina ela não pode mais ser interrompida.

Deseja-se encontrar uma seqüência de execução das operações das n tarefas nas 2 máquinas, ou seja, um escalonamento das operações, de modo que a soma dos tempos de término das operações em M_2 seja mínima. Logo, a função objetivo é:

$$\min f = \sum_{j=1}^n f_{2j}.$$

Note que esta função objetivo é equivalente a dizer que o que se busca é minimizar o tempo médio de conclusão das tarefas. Vale observar que trata-se um problema de **minimização** e, consequentemente, o limitante dual (primal) é um limite inferior (superior) para f . Na literatura este problema é conhecido pelo seu nome em inglês *Flowshop Scheduling Problem* e, por isso, será usada a notação FSP para denotá-lo.

Dentre os resultados conhecidos para o FSP destacam-se: (i) a versão de decisão de FSP é NP-completo, e (ii) existe um escalonamento ótimo no qual a sequência

de execução das tarefas é a mesma nas duas máquinas (conhecido por *escalonamento permutado*⁵) e no qual não há tempo ocioso *desnecessário* entre as tarefas. Este último resultado é particularmente importante pois permite que a busca por soluções ótimas fique restrita às $n!$ permutações das n tarefas, uma vez que isso define a ordem das operações nas máquinas e os seus instantes de início. O algoritmo *branch-and-bound* descrito a seguir está fundamentado neste fato.

Os conceitos anteriores são ilustrados a seguir. A tabela abaixo fornece os tempos de execução das operações para uma instância constituída de 3 tarefas.

t_{ij}	M_1	M_2
Tarefa 1	2	1
Tarefa 2	3	1
Tarefa 3	2	3

Para o escalonamento permutado $(2, 3, 1)$, a solução obtida é representada pelo esquema abaixo onde as linhas correspondem às máquinas e as colunas a unidades de tempo. Deste modo, na célula (i, j) está indicado o número da tarefa sendo executada na máquina M_i no instante de tempo j .

	1	2	3	4	5	6	7	8	9
M_1	2	2	2	3	3	1	1		
M_2				2	3	3	3	1	

O custo desta solução é $f = f_{21} + f_{22} + f_{23} = 9 + 4 + 8 = 21$. Para esta instância, com o será visto, a escalonamento permutado ótimo é dado por $(1, 3, 2)$:

	1	2	3	4	5	6	7	8	9
M_1	1	1	3	3	2	2	2		
M_2			1	3	3	3	3	2	

O custo da solução ótima é $f = f_{21} + f_{22} + f_{23} = 3 + 8 + 7 = 18$.

Como existe uma solução ótima que é um escalonamento permutado, uma solução viável pode ser representada apenas por uma permutação de n , do mesmo modo que no problema do caixeiro viajante. A partir desta observação, uma opção no projeto do algoritmo *branch-and-bound* é utilizar a mesma forma de particionamento do espaço de estados que foi adotada no Exemplo 3.8.2. Contudo, ainda é preciso definir como calcular limitantes duais (inferiores). Como dito anteriormente, isto pode ser alcançado mediante a resolução de relaxações do FSP. Duas relaxações são consideradas aqui.

Suponha que em um dado nó u da árvore de enumeração um subconjunto R de tarefas já tenham sido escalonadas, sendo $|R| = r$. Seja t_k , $k = 1, \dots, n$, o índice da k -ésima tarefa em qualquer escalonamento que possa ser representado por um nó na subárvore cuja raiz é o nó corrente u . O custo deste escalonamento será:

$$f = \sum_{i \in R} f_{2i} + \sum_{i \notin R} f_{2i}. \quad (15)$$

⁵Traduzido livremente do inglês “permutation schedule”.

Observe que, no momento da exploração do nó u , o primeiro termo desta soma já está definido, uma vez que ele corresponde as tarefas de R , cujas posições na permutação foram definidas nas ramificações executadas pelo algoritmo desde o nó raiz até atingir u . Ou seja, caberá ao algoritmo decidir ainda sobre a ordem das tarefas que não estão em R . Fica claro então que, para calcular um limitante dual em u , precisamos encontrar um minorante para a segunda somatória, o que pode ser atingido relaxando-se as restrições do FSP.

A primeira relaxação supõe que a segunda operação de cada tarefa em \bar{R} (i.e., fora de R) comece a ser executada na máquina M_2 imediatamente após o término da sua primeira operação em M_1 . Para uma ordem fixa das tarefas em \bar{R} , um minorante para a segunda soma da equação (15) seria:

$$S_1 = \sum_{k=r+1}^n [f_{1,t_r} + (n-k+1)t_{1,t_k} + t_{2,t_k}]. \quad (16)$$

Pode-se chegar à equação acima percebendo que, nesta relaxação, o tempo de término da tarefa k em M_2 é dado por $f_{1,t_r} + \sum_{l=1}^k t_{1,t_l} + t_{2,t_k}$. Pode-se notar que S_1 é um limitante inferior para S ao ver que esta relaxação admite que tarefas sejam executadas simultaneamente em M_2 .

A segunda relaxação é obtida ao se supor que cada tarefa em \bar{R} começa em M_2 imediatamente depois que a tarefa precedente termina sua execução naquela máquina. Nesta hipótese, a segunda operação de uma tarefa em \bar{R} poderia ser iniciada antes que a primeira operação da mesma tarefa estivesse concluída em M_1 , portanto, é claramente uma relaxação do FSP. Com isto, o tempo de término da k -ésima em \bar{R} seria a soma dos tempos de execução de todas as tarefas de \bar{R} que a precedem mais o tempo de conclusão da tarefa r (a última de R) em M_2 , i.e., f_{2,t_r} . Na verdade, alguma melhora pode ser obtida no limitante se, ao invés de f_{2,t_r} , for usado o valor $f_{1,t_r} + \min_{i \notin R}(t_{1,i})$, que seria o menor tempo de término possível de uma operação de uma tarefa de \bar{R} em M_1 . A partir daí, um minorante para a segunda soma da equação (15) seria:

$$S_2 = \sum_{k=r+1}^n [\max(f_{2,t_r}, f_{1,t_r} + \min_{i \notin R} t_{1,i}) + (n-k+1)t_{2,t_k}]. \quad (17)$$

Note que os minorantes obtidos nas equações (16) e (17) pressupõem uma ordem fixa das tarefas em \bar{R} . Porém, em cada um dos casos a ordem que leva ao menor valor da segunda soma da equação (15) pode ser computada facilmente. De fato, a minimização de S_1 é atingida ordenando-se as tarefas na ordem crescente dos valores de t_{1,t_k} enquanto a de S_2 é alcançada ordenando-se as tarefas na ordem crescente dos valores de t_{2,t_k} . Denotando-se por \hat{S}_1 e \hat{S}_2 os mínimos de S_1 e S_2 calculados como descrito acima, tem-se um *limitante inferior* para o FSP no nó corrente da árvore de enumeração dado por:

$$f \geq \sum_{i \in R} f_{2,i} + \max(\hat{S}_1, \hat{S}_2). \quad (18)$$

Usando estes limitantes duais e a forma de representação do espaço de estados sugerida, a árvore de enumeração resultante da aplicação do algoritmo *branch-and-bound* usando a estratégia do melhor limitante seria aquela vista na Figura 3.15. Para entender como se chegou a esta árvore, supõe-se que, se a solução, usualmente inviável, que corresponde aos limitantes \hat{S}_1 e \hat{S}_2 em um dado nó for viável, esta será usada para atualizar o limitante superior (primal). Ademais,

note-se que a ordenação das tarefas em ordem não decrescente do tempo de execução em M_1 (M_2), que é determinante para o cálculo de \hat{S}_1 (\hat{S}_2), é dada por $\{1, 3, 2\}$ ($\{1, 2, 3\}$). Dito isto, as soluções que levam aos limitantes \hat{S}_1 e \hat{S}_2 no nó 1, em que a primeira tarefa alocada é a tarefa 1 ($\equiv R = \{1\}$) são representadas por:

$$\begin{aligned}\hat{S}_1 : \quad & \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline M_1 & 1 & 1 & 3 & 3 & 2 & 2 & 2 \\ M_2 & & & 1 & & 3 & 3 & 3 & 2 \end{array} \implies f_{2,1} + \hat{S}_1 = 18, \\ \hat{S}_2 : \quad & \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline M_1 & 1 & 1 & (3) & (3) & & & & & \\ M_2 & & 1 & & 2 & 3 & 3 & 3 & 3 \end{array} \implies f_{2,1} + \hat{S}_2 = 16.\end{aligned}$$

A ociosidade observada em M_2 antes do início da tarefa 2 deve-se ao fato de que em M_1 , o tempo mínimo de espera até a conclusão da próxima tarefa (o qual seria alcançado se a tarefa 3 se sucedesse à tarefa 1) termina no instante 4. Note-se ainda que, no cálculo de \hat{S}_1 , a solução é viável e, consequentemente, ótima para aquela subárvore (limitantes primais e duais idênticos), forçando a poda deste nó por otimalidade. Para o nó 2 em que $R = \{2\}$, pode-se constatar que os cálculos de \hat{S}_1 e \hat{S}_2 corresponderiam às seguintes situações:

$$\begin{aligned}\hat{S}_1 : \quad & \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline M_1 & 2 & 2 & 2 & 1 & 1 & 3 & 3 & & & \\ M_2 & & & 2 & & 1 & & 3 & 3 & 3 \end{array} \implies f_{2,2} + \hat{S}_1 = 20, \\ \hat{S}_2 : \quad & \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline M_1 & 2 & 2 & 2 & (1) & (1) & & & & \\ M_2 & & & 2 & & 1 & 3 & 3 & 3 \end{array} \implies f_{2,2} + \hat{S}_2 = 19.\end{aligned}$$

Como o nó 1 já deu uma solução primal de valor 18, menor do que o limite dual de 20 computado para o nó 2, este último é podado por limite⁶. Por fim, no nó 3 em que $R = \{3\}$, as situações associadas aos limitantes duais \hat{S}_1 e \hat{S}_2 seriam aquelas vistas abaixo:

$$\begin{aligned}\hat{S}_1 : \quad & \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline M_1 & 3 & 3 & 1 & 1 & 2 & 2 & 2 \\ M_2 & & 3 & 3 & 3 / 1 & & & 2 \end{array} \implies f_{2,3} + \hat{S}_1 = 18, \\ \hat{S}_2 : \quad & \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline M_1 & 3 & 3 & (1) & (1) & & & & \\ M_2 & & 3 & 3 & 3 & 1 & 2 \end{array} \implies f_{2,3} + \hat{S}_2 = 18.\end{aligned}$$

Perceba que, no cálculo de \hat{S}_1 há uma sobreposição das tarefas 1 e 3 no instante 5 em M_2 , o que é permitido nesta relaxação. Note ainda que haverá a execução simultânea da mesma tarefa na solução de \hat{S}_2 . De qualquer maneira, estes limitantes não impedem a poda do nó 3 por limite, uma vez que o limite primal do nó 1 já é igual a 18. Logo, não existem mais nós ativos na árvore de enumeração e o valor ótimo retornado é 18.

⁶Alternativamente poder-se-ia pensar também em podar este nó por otimalidade dado que o cálculo de \hat{S}_1 conduz à uma solução viável.

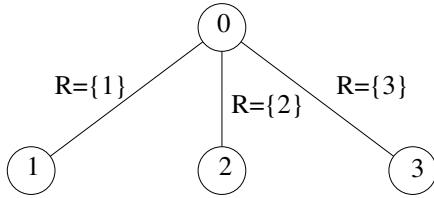


Figura 3.15. Aplicação do algoritmo *branch-and-bound* ao FSP.

3.8.3. Algoritmo *Branch-and-bound* para Programação Linear Inteira

Como mencionado na Seção 3.6, diversos problemas de otimização podem ser modelados usando Programação Linear e adicionando-se a restrição de que algumas ou todas as variáveis tomem valores inteiros (as chamadas *restrições de integralidade*). Ao fazer isso, chega-se a um problema de Programação Linear Inteira. Diferentemente do PL que pertence à classe \mathbb{P} , a PLI está em NP-difícil . Por isso, os problemas de PLI são candidatos a serem resolvidos por um algoritmo de *branch-and-bound*. Neste caso, o limitante dual pode ser computado através da relaxação linear do PLI, obtida ao se remover as restrições de integralidade. Quanto à ramificação da árvore de enumeração, uma técnica comumente empregada é escolher a variável “mais fracionária” na solução ótima da relaxação linear para gerar os filhos do nó corrente. Ela funciona do seguinte modo. Suponha que em um nó da árvore tenha sido resolvida a relaxação linear, obtendo-se uma solução ótima $x^* \in \mathbb{R}^n$ em que pelo menos uma variável x_i^* tem parte fracionária não nula, ou seja, $x_i^* = \lfloor x_i^* \rfloor + f_i$, com $0 < f_i < 1$ para algum $i = 1, \dots, n$. Quando isso ocorre, seleciona-se a variável x_j em que f_j se aproxima mais do valor 0,5 (meio) e faz-se uma ramificação em que, o filho à esquerda do nó corrente representará o conjunto das soluções em que $x_j \leq \lfloor x_j^* \rfloor$, enquanto o filho da direita está associado ao conjunto das soluções em que $x_j \leq \lceil x_j^* \rceil$. Novamente, o critério de escolha do próximo nó ativo a ser ramificado pode obedecer à estratégia do *melhor limitante (dual)* como nos exemplos anteriores.

Para se ter uma ideia do funcionamento do algoritmo, será usado o mesmo PL do exemplo da Figura 3.7 ao qual se acrescenta a restrição de que as variáveis devam ser inteiras. Ou seja, a relaxação linear deste problema é dada por

$$(PL0) \quad \max z = 2x_1 + 3x_2 \quad (19)$$

$$\text{s. a} \quad 2x_1 + 6x_2 \leq 15, \quad (20)$$

$$2x_1 - 2x_2 \leq 3, \quad (20)$$

$$x_1, x_2 \geq 0. \quad (21)$$

Na Figura 3.16 os círculos pretos representam as soluções inteiras viáveis do PLI enquanto a região hachurada corresponde à região de viabilidade de sua relaxação linear. Foi visto na Seção 3.6 que a solução ótima deste problema está no ponto extremo de coordenadas $(x_1^*, x_2^*) = (3, \frac{3}{2})$ (círculo branco). Adotando-se a regra de ramificação discutida anteriormente, como x_2^* é a única variável com parte fracionária não nula, o nó raiz da árvore de enumeração (associado a *PL0*), dará origem a dois filhos: à esquerda onde será exigido que $x_2 \leq 1$ e à direita onde as soluções deverão atender a $x_2 \geq 2$. As relaxações lineares dos PLIs correspondentes aos dois

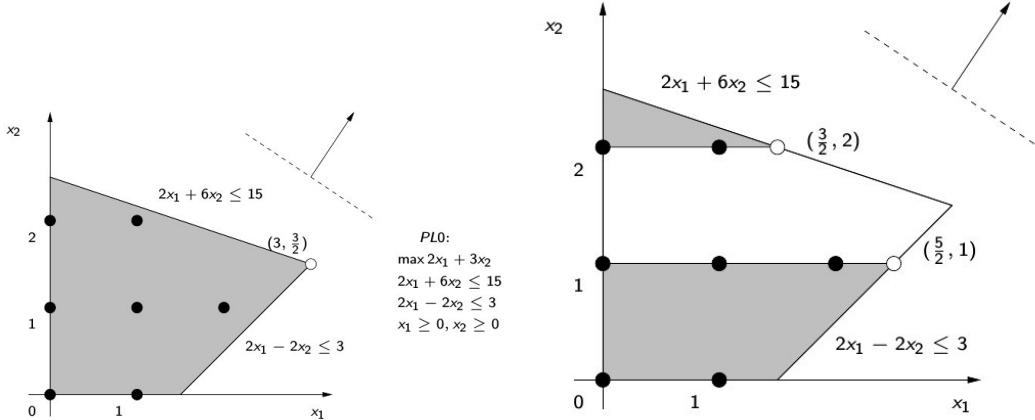


Figura 3.16. Aplicação do algoritmo *branch-and-bound* a um PLI: relaxação linear na raiz.

$$PL1 = PL0 + \{x_2 \leq 1\}, (z_1 = 8)$$

$$PL2 = PL0 + \{x_2 \geq 2\}, (z_2 = 9)$$

Figura 3.17. Aplicação do algoritmo *branch-and-bound* a um PLI: relaxações lineares nos filhos da raiz.

filhos estão identificadas pelas regiões hachuradas da Figura 3.17, sendo o quadrilátero associado ao filho esquerdo e o triângulo ao filho direito. Denota-se por $PL1$ e $PL2$, respectivamente, as relaxações lineares dos filhos esquerdo e direito da raiz. A solução ótima de $PL1$ é o ponto $(\frac{5}{2}, 1)$ e a de $PL2$ $(\frac{3}{2}, 2)$ como se observa na figura. Se o algoritmo *branch-and-bound* for executado até o final, a parte da árvore de enumeração explorada será aquela mostrada na Figura 3.18. Em cada nó árvore vê-se os valores de x_1 e x_2 na solução ótima da relaxação assim como o limitante dual produzido por ela. Note que pelo critério do *melhor limitante*, todos nós da subárvore à direita da raiz serão explorados antes do seu filho esquerdo (à exceção, obviamente, daqueles que levam a inviabilidades). Com isso, o filho esquerdo acaba sendo podado *por limitante*, uma vez que o nó mais à esquerda no último nível da árvore, que foi podado *por optimidade*, gera um limitante primal de valor 8, igual ao limitante dual do filho esquerdo da raiz. As Figuras 3.19 e 3.20 mostram graficamente as demais relaxações resolvidas durante o percurso na árvore.

3.9. Considerações Finais

O texto apresenta algumas das principais técnicas usadas no desenvolvimento de algoritmos para problemas de Otimização Combinatória e seu conteúdo foi guiado pelo material usado pelos autores para ministrar as disciplinas de *Projeto e Análise de Algoritmos* nos cursos de *Ciência da Computação* e *Engenharia da Computação* na Universidade Estadual de Campinas.

Por ser introdutório e por restrições de espaço, várias técnicas e abordagens importantes não puderam ser vistas. Assim, optou-se pela abordagem exata, para a qual muitos problemas de otimização puderam ser contemplados e resolvidos por técnicas importantes da área. Dentre

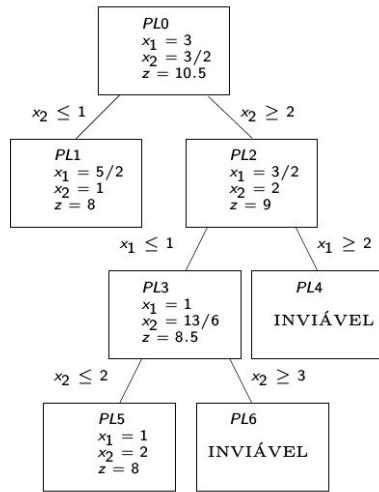


Figura 3.18. Aplicação do algoritmo *branch-and-bound* a um PLI: porção explorada da árvore de enumeração.

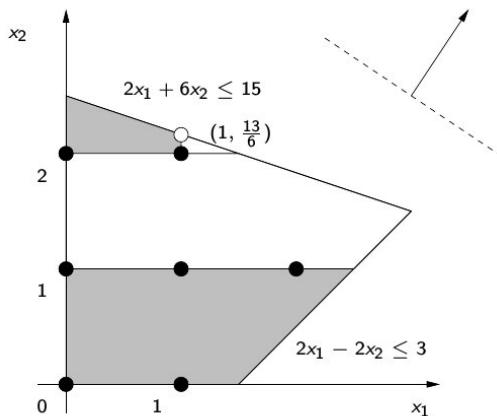


Figura 3.19. Aplicação do algoritmo *branch-and-bound* a um PLI: relaxações lineares intermédiárias (Parte I).

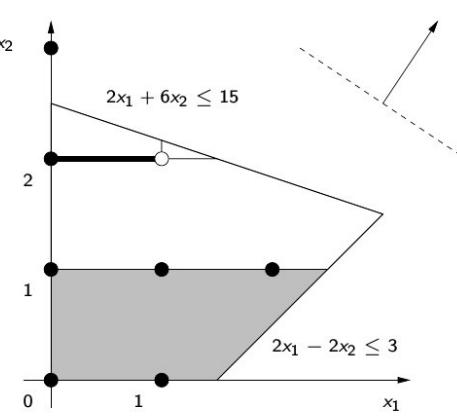


Figura 3.20. Aplicação do algoritmo *branch-and-bound* a um PLI: relaxações lineares intermédiárias (Parte II).

os aspectos teóricos que não pudemos explorar, mas são importantes no desenvolvimento de algoritmos da área, são os aspectos relativos à complexidade computacional, principalmente as classes de complexidade NP -completo e NP -difícil, sendo que esta última contempla inúmeros problemas práticos. O estudo destas classes de complexidade permite entender a dificuldade de se desenvolver algoritmos de tempo polinomial para tais problemas, bem como técnicas para reconhecê-los. Em particular, a menos que a improvável conjectura $\mathbb{P} = \text{NP}$ seja válida, não existirão algoritmos de tempo polinomial para encontrar soluções ótimas de tais problemas [Garey e Johnson 1979]. Assim, é de fundamental importância conhecer a complexidade computacional dos problemas, uma vez que estas podem guiar na escolha das técnicas e abordagens a serem usadas no desenvolvimento de algoritmos para cada problema.

Dentre outras técnicas consideradas importantes no desenvolvimento de algoritmos exatos, citamos a técnica *branch-and-cut*, que leva a bons resultados práticos na resolução de vários problemas. Em particular, estão entre os melhores métodos para obter algoritmos exatos para problemas envolvendo restrições de conectividade para problemas NP -difíceis, como o Problema do Caixeiro Viajante. Dentre as abordagens importantes que não foram exploradas no texto, temos as de *Algoritmos de Aproximação* e de *Heurísticas*. A ideia básica ao desenvolver *Algoritmos de Aproximação* é sacrificar a busca por uma solução ótima em favor de uma solução com valor próximo do ótimo, utilizando algoritmos de tempo polinomial, em contraposição aos algoritmos de tempo exponencial. *Heurísticas* são procedimentos que buscam por soluções de boa qualidade, dentro das limitações dos recursos existentes para sua obtenção. As heurísticas não necessariamente dão garantias de se encontrar soluções ótimas, ou mesmo de se encontrar uma solução que atenda aos requisitos estruturais do problema.

Agradecimentos: Os autores agradecem a Profa. Cláudia Linhares pelo apoio que tiveram para desenvolver este texto de maneira mais abrangente, o revisor pelas anotações e observações para a melhoria do texto, e ao CNPq pelo suporte financeiro.

Referências bibliográficas

- [Ahuja et al. 1993] Ahuja, R. K., Magnanti, T. L. e Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [Applegate et al. 2007] Applegate, D. L., Bixby, R. E., Chvatal, V. e Cook, W. J. (2007). *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA.
- [Bazaraa et al. 2009] Bazaraa, M. S., Jarvis, J. J. e Sherali, H. D. (2009). *Linear Programming and Network Flows*. Wiley-Interscience. 4th edition.
- [Bondy e Murty 2008] Bondy, J. e Murty, U. (2008). *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag. 654 pgs.
- [Brassard e Bratley 1988] Brassard, G. e Bratley, P. (1988). *Algorithmics: Theory and Practice*. Prentice-Hall.
- [Cormen et al. 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L. e Stein, C. (2009). *Introduction to Algorithms (3. ed.)*. MIT Press.

- [Dasgupta et al. 2008] Dasgupta, S., Papadimitriou, C. H. e Vazirani, U. V. (2008). *Algorithms*. McGraw-Hill.
- [Erickson 2015] Erickson, J. (Acessado em Maio de 2015). Algorithms and models of computation. <http://www.cs.illinois.edu/~jeffe/teaching/algorithms>.
- [Feofiloff et al. 2004] Feofiloff, P., Kohayakawa, Y. e Wakabayashi, Y. (2004). Uma introdução sucinta à teoria dos grafos. <http://www.ime.usp.br/~pf/teoriadosgrafos/>. Texto da II Bienal da SBM.
- [Ferreira e Wakabayashi 1996] Ferreira, C. E. e Wakabayashi, Y. (1996). *Combinatória Poliedrica e Planos-de-Corte Faciais*. 10^a Escola de Computação, Campinas.
- [Garey e Johnson 1979] Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.
- [Karmarkar 1984] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395.
- [Khachiyan 1979] Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096.
- [Kloks 2012] Kloks, T. (2012). *Advanced Graph Algorithms*.
- [Manber 1989] Manber, U. (1989). *Introduction to algorithms - a creative approach*. Addison-Wesley.
- [Nemhauser e Wolsey 1988] Nemhauser, G. L. e Wolsey, L. A. (1988). *Integer and combinatorial optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley.
- [Papadimitriou e Steiglitz 1982] Papadimitriou, C. H. e Steiglitz, K. (1982). *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [Rosen 2012] Rosen, K. H. (2012). *Discrete Mathematics and Its Applications: And Its Applications*. McGraw-Hill Higher Education, sétima edição.
- [Schrijver 1986] Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley, Chichester.
- [Szwarcfiter 1988] Szwarcfiter, J. (1988). *Grafos e Algoritmos Computacionais*. Editora Campus Ltda, second edição.
- [Wolsey 1998] Wolsey, L. A. (1998). *Integer programming*. Wiley-Interscience, New York, NY, USA.
- [Ziviani 2011] Ziviani, N. (2011). *Projetos de Algoritmos com implementações em Pascal e C*. Cengage Learning.

Capítulo

4

IFML: Linguagem de Modelagem de Fluxo de Intereração¹

Raul Sidnei Wazlawick

Abstract

This chapter shows how to design front-end interfaces for information systems using a hypertext specification language known as IFML: Interaction Flow Modeling Language. Initially, the chapter presents the information view components of IFML, which are used in connection to the design class diagrams to manage the presentation of the information. The integration of view components and flows into pages and areas is also presented, as well as some interface design patterns. Operation components may also be used in connection to view components in order to define basic operations over objects and to connect the view to the operations implemented by the controller. The chapter concludes with the presentation of a CRUD interface modeled in IFML and a high-level design for a system use case.

Resumo

Este capítulo mostra como projetar uma interface com usuário para sistemas de informação usando uma linguagem de especificação de hipertexto conhecida como IFML: Interaction Flow Modeling Language. Inicialmente, os componentes de informação da IFML são apresentados, os quais são usados em conexão com diagramas de classe de design para gerenciar a apresentação de informação. A integração dos componentes de visão e fluxos em páginas e áreas é também apresentada, bem como alguns padrões de design de interface. Componentes de operação podem ser usados em conexão com os componentes de visão para definir operações básicas sobre objetos, bem como associar a visão com operações implementadas pelo controlador. O capítulo termina com a apresentação de uma interface CRUD modelada em IFML e um design em alto nível para um caso de uso de sistema.

¹ Este texto é uma adaptação do capítulo 12 do livro “Análise e Design Orientados a Objetos para Sistemas de Informação: Modelagem com UML, OCL e IFML”, do mesmo autor, publicado pela Elsevier em 2015 e também disponível em inglês com o título “Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML”, publicado pela Morgan Kaufmann em 2014.

4.1. Introdução ao Design da Camada de Interface

Sistemas de software, especialmente aplicações voltadas para sistemas de informação, usualmente são projetadas em camadas, de forma que diferentes preocupações sejam abordadas em diferentes partes do design. O modelo de três camadas (Eckerson, 1995) é, assim, bastante popular: ele separa a arquitetura do sistema nas camadas de *apresentação* (interface com usuário), *domínio* (lógica de transformação de dados) e *persistência* (armazenamento de dados).

O processo de análise e design de sistemas orientados a objetos, em linhas gerais, parte de uma análise de requisitos a partir da qual pode-se inicialmente definir a camada de domínio como um modelo usualmente representado por um diagrama de classes (Miles & Hamilton, 2006). A camada de persistência normalmente é derivada desse modelo já que, via de regra, cada classe que represente informação de domínio (entidade) será representada, por exemplo, através de uma tabela relacional, embora outras formas sejam possíveis. Como o mapeamento entre objetos e tabelas relacionais nem sempre é tão direto (Ireland *et al.*, 2009), os responsáveis pelo design de banco de dados acabam usando diagramas como o Entidade-Relacionamento (Chen, 1976) para representar as tabelas que vão conter as informações gerenciadas pelo sistema. Várias ferramentas CASE (Computer Aided Software Engineering) já permitem há muitos anos a geração de código a partir destes diagramas, facilitando, assim, o trabalho dos programadores.

Já em relação à camada de apresentação, a abordagem usual acaba sendo baseada no desenho de interfaces acompanhado de comentários que explicam como cada janela ou tela deve se comportar. Eventualmente, o fluxo de interação do usuário com a interface acaba sendo modelado com diagramas de atividades, máquina de estados ou mesmo fluxogramas.

Mais recentemente, porém, sentiu-se a necessidade de definir um diagrama específico para modelagem de fluxos de interação entre usuário e computador, o qual pudesse, de forma clara e sintética, representar os diferentes componentes de uma interface com usuário e como os dados transitam entre esses componentes (Object Management Group, 2015). Além disso, seria necessário também que tal diagrama permitisse associar os eventos de interface, como, por exemplo, o pressionar de um botão, com chamadas a operações de sistema implementadas na camada de domínio e usualmente encapsuladas por uma classe controladora-fachada (Gamma *et al.*, 1995).

A partir dessa necessidade é que foi desenvolvida a Linguagem de Modelagem de Fluxo de Intereração ou *Interaction Flow Modeling Language* (IFML), a qual é uma linguagem compatível com a UML (*Unified Modeling Language*) que pode ser usada para modelar interfaces com usuário nas quais há uso intensivo de dados. Ela é uma evolução de WebML (Ceri, *et al.*, 2003) e foi adotada como padrão pela OMG em 2015 (Object Management Group, 2015).

Este capítulo tem como objetivo apresentar os conceitos mais fundamentais de IFML e seu potencial de modelagem. Mais informação pode ser encontrada no site oficial da linguagem² e no livro de Brambilla e Frernali (2014). A notação usada neste capítulo corresponde à usada na ferramenta WebRatio®, a qual é a primeira a implementar o padrão.

² A versão 1.0 da especificação oficial de IFML foi publicada em março de 2015. Disponível em: <http://www.omg.org/spec/IFML/1.0/PDF/>.

³ <http://www.webratio.com>

A Seção 4.2 apresenta uma visão geral dos diferentes modelos associados com o uso da linguagem. A Seção 4.3 apresenta os componentes de visão, ou seja, os elementos que permitem especificar que dados serão apresentados ou coletados em uma interface gráfica. A Seção 4.4 introduz os conceitos de organização de um *site* com o uso de contêineres, ou seja, áreas e páginas. A Seção 4.5 apresenta os tipos de *fluxos* que podem ser definidos para navegação entre páginas e componentes de visão, bem como para obtenção de dados. A Seção 4.6 apresenta alguns padrões de interface *Web* bastante comuns em aplicações reais. A Seção 4.7 apresenta os componentes de operação que permitem especificar transformações nos dados, bem como conectar os componentes de visão com operações de sistema a serem executadas na camada de domínio. A Seção 4.8 apresenta a especificação completa de um CRUD (*Create, Retrieve, Update e Delete*) com IFML. A Seção 4.9 mostra como modelar uma interface com usuário em IFML a partir de um caso de uso expandido. A Seção 4.10 apresenta as considerações finais do capítulo.

4.2. A Linguagem

IFML é voltada para modelagem de interfaces, especialmente aquelas que fazem uso intensivo de dados, tais como sistemas de informação. O desenvolvimento de aplicações com IFML considera que pelo menos cinco modelos sejam usados:

- *Modelo estrutural*: ele representa as estruturas e a organização dos dados. Ceri *et al.* (2003) utilizam o diagrama entidade-relacionamento para definir o modelo estrutural. Entretanto, o diagrama de classes de UML e outros podem ser usados com os mesmos propósitos.
- *Modelo de derivação*: ele foi originalmente proposto para modelar dados que podem ser calculados a partir de outros dados. O modelo de derivação pode ser entendido como o conjunto de definições de atributos e associações derivadas que usualmente são definidos no diagrama de classes e podem ser escritos, por exemplo, com a linguagem OCL (Object Management Group, 2010).
- *Modelo de composição*: ele inclui a definição de contêineres como páginas (Seção 4.4.1) e áreas (Seção 4.4.2) que agregam os de componentes de visão mais básicos (Seção 4.3).
- *Modelo de navegação*: ele permite definir os fluxos de navegação entre as páginas e componentes de visão (Seção 4.5.1).
- *Modelo de apresentação*: ele permite definir o posicionamento dos componentes de visão da interface e sua aparência.

Este capítulo trata particularmente dos modelos de *composição* e *navegação*. Os modelos estrutural e de derivação podem ser encontrados nos trabalhos de Ceri *et al.* (2003) ou Wazlawick (2015). Para o modelo de apresentação, a ferramenta WebRatio®⁴ permite o uso de planilhas de estilo XSL (*Extensible Stylesheet Language – Linguagem Extensível de Folha de Estilos*)⁵. A ferramenta de geração de código usa essas regras de apresentação para produzir páginas de acordo com o design produzido com outras ferramentas. Para os exemplos apresentados neste capítulo, são mostradas as interfaces padrão renderizadas pela ferramenta sem o uso de estilos ou design especial. Mesmo os

⁴ A primeira ferramenta compatível com IFML.

⁵ Disponível em: <www.w3.org/Style/XSL/>. 192

nomes dos campos foram deixados idênticos aos nomes dos atributos das classes com o objetivo de promover a compreensão mais do que a qualidade visual da apresentação.

4.3. Componentes de Visão

Componentes de visão são os elementos mais básicos em uma especificação IFML. Componentes de visão podem ser associados a classes de design e permitem a visualização de instâncias dessas classes. Componentes de visão em IFML podem ser especializados. Alguns exemplos de especializações de componentes de visão são:

- *Detalhes*: mostra informação sobre um único objeto.
- *Detalhes múltiplos*: mostra informação sobre uma coleção de instâncias da mesma classe.
- *Lista simples*: mostra múltiplas instâncias de uma classe na forma de uma lista.
- *Lista*: um melhoramento da lista simples que permite *scrolling* e ordenação dinâmica.
- *Lista de múltipla escolha*: uma lista que permite seleção múltipla.
- *Hierarquia*: mostra múltiplas instâncias de diferentes classes organizadas em uma hierarquia, tal como todo-parte ou mestre-detalhe (Seffah, 2015).
- *Hierarquia recursiva*: mostra múltiplas instâncias de uma única classe organizadas hierarquicamente, como no caso de estruturas organizacionais.
- *Scroller*: permite as operações típicas de *scroll* sobre uma sequência de objetos.
- *Formulário*: permite entrada de dados baseada em formulário.
- *Formulário múltiplo*: similar ao formulário, mas permite a edição de múltiplos objetos de cada vez.
- *Mensagem*: permite imprimir mensagens em uma página.
- *Calendário*: mostra um calendário perpétuo.

Existem outros componentes de visão e novos componentes podem ser definidos pelo usuário usando estereótipos (Brambilla & Frernali, 2014). Este capítulo apresenta informações sobre os mais fundamentais dentre eles. Para cada tipo de componente de visão uma definição gráfica é apresentada, bem como a possível aparência da página renderizada por WebRatio®. Todos os exemplos são baseados no diagrama de classes de design (DCD) da Figura 4.1.

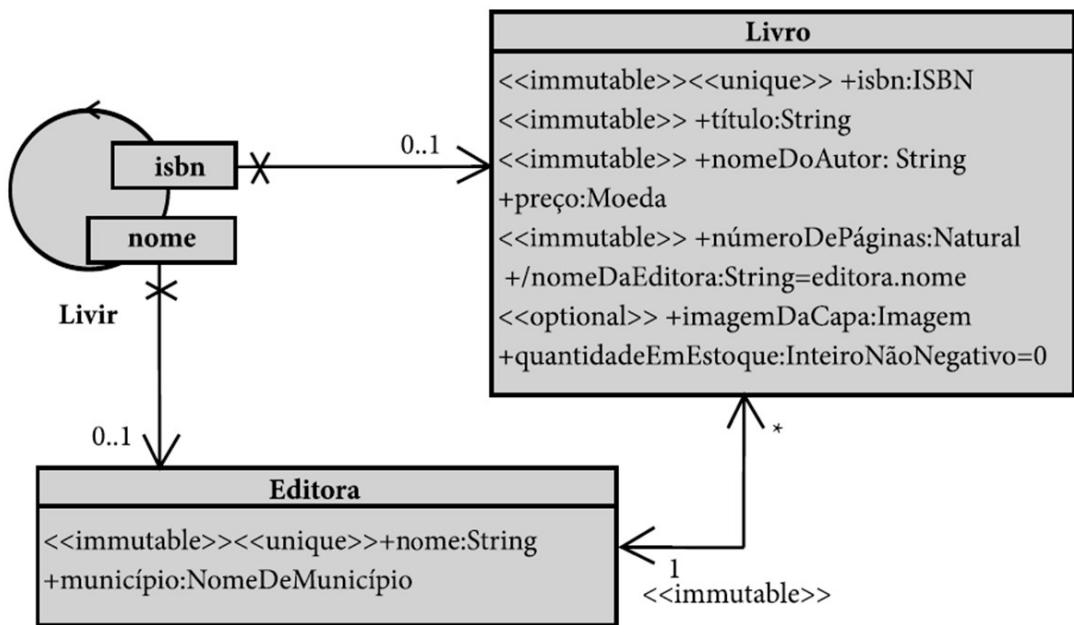


Figura 4.1. Diagrama de classes de design de referência.

Na Figura 4.1, as classes *Editora* e *Livro* representam conceitos ou entidades e a classe *Livr* (Livraria Virtual) é uma classe controladora de sistema, ou seja, ela é fachada que encapsula os objetos em relação à camada de interface. O estereótipo *<<immutable>>* significa que o atributo ou associação depois de definido não pode mais ser modificado, *<<unique>>* significa que o atributo não pode ter seu valor repetido em diferentes instâncias da classe e *<<optional>>* significa que o atributo pode ter valor indefinido (todos os demais atributos devem ser obrigatoriamente definidos desde a criação do objeto ou ele estará inconsistente). A barra “/” indica que o atributo ou papel de associação é *derivado*, ou seja, obtido através de um cálculo.

4.3.1. Componente de Visão *Detalhes*

O componente de visão *detalhes* apresenta informação sobre um único objeto de uma dada classe. Ele é definido pelas seguintes propriedades:

- Um *nome* que é escolhido pelo designer.
- Uma *entidade* ou *fonte de dados* que é usualmente uma classe do diagrama de classes de design.
- *Atributos exibidos* que é uma lista de atributos da classe a serem exibidos quando o componente *detalhes* é renderizado.

A maioria dos componentes de visão como *detalhes* também permite a definição de uma ou mais *expressões condicionais* que funcionam como seletores ou filtros sobre o conjunto de instâncias a serem exibidas. Existem três tipos de expressões condicionais:

- *Condição-chave*: cada objeto no modelo de dados é identificado por um OID (*Object Identifier*) – um atributo que é único, obrigatório e imutável. Ele não deve ser confundido com nenhum dos atributos da classe conceitual, mesmo os estereotipados com *<<unique>>* e *<<immutable>>*, tais como ISBN ou CPF. O OID é um código gerado internamente e corresponde à chave primária de um objeto. OIDs não podem ser conhecidos ou atualizados pelo usuário. Uma

condição-chave permite que sejam selecionadas uma ou mais instâncias de uma dada classe baseando-se no valor do seu atributo OID.

- *Condição de atributo:* ela permite a seleção de uma ou mais instâncias baseando-se nos valores de seus atributos. Operações como *maior que*, *igual* ou *contém* podem ser usadas para comparar o valor de um atributo com um dado parâmetro.
- *Condição de papel:* ela permite a seleção de uma ou mais instâncias baseando-se nas suas ligações.

A Figura 4.2 exibe em seu lado esquerdo um componente de visão *detalhes* que deve mostrar dados sobre uma instância de um livro para um dado ISBN. A expressão condicional *isbn=0517149257* determina qual instância é mostrada. Se esta instância não existisse, então nenhuma informação seria renderizada por aquele componente. Como o ISBN é um atributo que é único e obrigatório, é seguro assumir que não mais do que um livro é apresentado por este componente de visão detalhes. O lado direito da figura mostra uma possível renderização Web para o componente de visão detalhes.



Figura 4.2. Um componente de visão detalhes com uma condição de atributo simples e sua renderização.

A renderização final da página pode variar porque é possível adicionar definições de estilo de forma que interfaces com diferentes design e aparência possam ser produzidas.

Se o componente de visão detalhes da Figura 4.2 for mudado para permitir que apenas os atributos *título* e *nomeDoAutor* apareçam, então ele produziria o resultado renderizado na Figura 4.3.

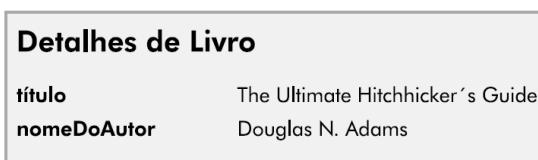


Figura 4.3. Uma possível renderização de um componente de visão detalhes com um número menor de atributos selecionados.

Usualmente, a seleção dos atributos a serem mostrados não é indicada graficamente no diagrama IFML. Com o uso de uma ferramenta de edição, essa informação fica oculta nos próprios componentes de visão e pode ser consultada quando se exibe informações internas sobre o componente.

4.3.2. Componente de Visão *Detalhes Múltiplos*

Detalhes múltiplos apresenta um grupo de instâncias de uma única classe de uma vez. Além das três propriedades já mostradas para o componente de visão detalhes, ele adiciona as seguintes propriedades:

- *Atributos de ordenação*: são os atributos usados para ordenar as instâncias.
- *Máximo de resultados*: especifica o número máximo de instâncias que podem ser recuperadas. Por *default*, todas as instâncias da classe são recuperadas.
- *Distinto*: especifica que elementos duplicados não serão mostrados mais do que uma vez.

O componente de visão detalhes múltiplos também permite a definição e o uso de expressões condicionais para selecionar quais elementos devem ser mostrados. A Figura 4.4 apresenta um exemplo de detalhes múltiplos e sua renderização considerando que apenas três livros satisfazem a expressão condicional e que a ordenação acontece pelo título.

The screenshot shows a component visualization titled "Detalhes Múltiplos de Livros". It features a small icon of a book and the word "Livro" below it. A condition expression "[nomeDoAutor = Douglas N. Adams]" is shown. Below this, a table titled "Detalhes Múltiplos de Livros" lists three books:

isbn	título	nomeDoAutor	número DePáginas	preço	imagem DaCapa	quantidade EmEstoque
0671746723	Dirk Gently's Holistic Detective Agency	Douglas N. Adams	306	6.99		12
0671742515	The Long Dark Tea-Time of the Soul	Douglas N. Adams	307	6.99		4
0517149257	The Ultimate Hitchhiker's Guide	Douglas N. Adams	815	122		4

Figura 4.4. Um componente de visão detalhes múltiplos e sua renderização.

Em oposição ao componente de visão detalhes que mostra apenas uma instância de cada vez, *detalhes múltiplos* apresenta todos os objetos que satisfazem a expressão condicional.

Se for necessário usar mais do que um atributo para determinar os objetos a serem apresentados pelo componente, então uma combinação de expressões condicionais pode ser usada. Essa combinação pode usar os operadores *and* e *or*. A Figura 4.5 apresenta um componente de visão detalhes múltiplos que mostra instâncias que têm *númeroDePáginas* maior do que 100 e *preço* menor do que R\$ 150,00.

A ferramenta usada para gerar o elemento apenas mostra a lista de condições usadas, sem especificar se elas são combinadas com *and* ou *or*. Para consultar essa informação é necessário abrir o componente e verificar sua especificação.



Figura 4.5. Um componente de visão detalhes múltiplos com uma expressão de condição composta.

4.3.3. Componente de Visão *Lista Simple*

O componente de visão *lista simple* apresenta um ou mais atributos de um conjunto de instâncias de uma classe na forma de uma lista. Enquanto o componente *detalhes múltiplos* é normalmente usado para visualizar os detalhes de vários objetos em uma mesma página, a lista simples e suas variações são usadas quando uma lista ou menu são necessários para o usuário selecionar um ou mais elementos e realizar ações com eles. As propriedades das listas simples são as mesmas do componente de visão detalhes múltiplos.

É possível, por exemplo, definir listas simples para selecionar livros com base em seus títulos. A Figura 4.6 apresenta o componente de visão para uma lista simples de títulos de livros e sua possível renderização. A ausência de uma expressão condicional no componente de visão de lista simples implica que os títulos de todas as instâncias de *Livro* serão mostrados.



Figura 4.6. Lista Simples e sua renderização.

4.3.4. Componente de Visão *Lista*

O componente de visão *lista* é uma versão aprimorada da lista simples com operações de *scroll* e reordenação dinâmica dos elementos a partir de um simples clique no cabeçalho da respectiva coluna de atributo. Além das propriedades já definidas para listas simples, ele inclui as seguintes propriedades específicas:

- *Ordenável*: um atributo booleano que quando é verdadeiro define que a lista pode ser ordenada dinamicamente.

- *Atributos de ordenação default*: se a lista for ordenável, define os atributos de ordenação que podem ser selecionados pelo usuário para ordenar a lista e também o critério default de ordenação quando a lista é renderizada pela primeira vez.
- *Tamanho do histórico de ordenação*: especifica quantas ações de ordenação do usuário devem ser lembradas pela lista. Assim, por exemplo, se o usuário ordenar pela coluna 3, depois pela coluna 1 e depois pela coluna 5, se apenas uma ação de ordenação for lembrada, ele poderá desfazer a última ordenação (pela coluna 5) e retornar à configuração com a ordenação pela coluna 1, mas não poderá mais retornar para a ordenação pela coluna 3 com a ação de desfazer.
- *Assinalável*: se verdadeiro, uma *checkbox* é renderizada para cada elemento da lista.
- *Fator de bloco*: especifica quantos elementos são mostrados de cada vez. Se não foi especificado ou se definido como -1, todos os elementos são mostrados. Se o fator de bloco for definido, então os comandos de *scrolling* são renderizados para a lista.

A Figura 4.7 mostra o componente de visão IFML para uma lista e sua renderização. Esta lista é definida para a classe *Livro* com três atributos: *nomeDoAutor*, *título* e *preço*. O fator de bloco é definido como 3.

Observe no topo da renderização na Figura 4.7 os comandos de *scroll* para a lista do primeiro até o último grupo de três elementos. Neste momento, os primeiros três elementos estão sendo mostrados, ordenados pelo seu título.

The figure shows the IFML definition of a list component and its rendered view. The IFML definition on the left is:

```

Lista de Livros
-----
[■] [■] [■] [■]
Livro

```

The rendered view on the right is:

Lista de Livros

first previous 1 to 3 of 8 next last
jump to 1 2 3

<u>título</u>	<u>nomeDoAutor</u>	<u>preço</u>
Dirk Gently's Holistic Detective Agency	Douglas N. Adams	6.99
Foundation and Empire	Isaac Asimov	5.99
Rama II	Arthur C. Clarke	6.99

Figura 4.7. Uma lista e sua possível renderização.

Se o usuário clicar em uma das ligações no topo de cada coluna, então os elementos da lista serão rearranjados e ordenados por aquela coluna. Se o usuário clicar, por exemplo, em *nomeDoAutor*, as linhas serão ordenadas pelos nomes dos autores.

4.3.5. Componente de Visão *Lista de Seleção Múltipla*⁶

A *lista de seleção múltipla* permite que o usuário selecione um conjunto de elementos de uma lista. Fora isso, ela é muito similar a uma lista simples. A Figura 4.8 apresenta a definição IFML de uma lista múltipla e sua renderização.

⁶ *Checkable list*.



The screenshot shows a Java Swing application window titled "Lista de Seleção Múltipla de Livros". On the left, there is a sidebar with the same title and four checkboxes. Below the checkboxes is a button labeled "Livro". The main area contains a table titled "Lista de Seleção Múltipla de Livros" with three columns: "título", "nomeDoAutor", and "preço". The table lists eight books, each with a checkbox next to its title. The first two books have checked checkboxes, while the others are unchecked.

título	nomeDoAutor	preço
<input type="checkbox"/> Shave the Wales	Scott Adams	6.99
<input checked="" type="checkbox"/> The Revenge of the Baby-Sat	Bill Watterson	8.95
<input type="checkbox"/> Foundation and Empire	Isaac Asimov	5.99
<input checked="" type="checkbox"/> Rama II	Arthur C. Clarke	6.99
<input checked="" type="checkbox"/> The Hammer of God	Arthur C. Clarke	6.99
<input type="checkbox"/> The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99
<input type="checkbox"/> Dirk Gently's Holistic Detective Agency	Douglas N. Adams	6.99
<input type="checkbox"/> The Ultimate Hitchhiker's Guide	Douglas N. Adams	122

Figura 4.8. Lista de seleção múltipla e sua renderização.

Se um evento for associado à seleção de múltiplos objetos então a lista também renderizará um botão para ativar o evento.

4.3.6. Componente de Visão *Formulário*

Um *formulário* é usado para entrada de dados. Ele é muito útil para a criação de novas instâncias e também para fornecer parâmetros para pesquisas, consultas e comandos.

Um formulário é composto por um conjunto de *campos*. Cada campo contém um valor alfanumérico. Existem campos textuais, campos de seleção e campos de seleção múltipla, embora outros possam ser definidos.

Um formulário pode ser ou não associado a uma classe. Se ele for associado a uma classe, então seus campos podem ser associados aos atributos da classe.

Formulários que são usados para coletar parâmetros para uma consulta ou comando usualmente não precisam ser associados a classes. Por exemplo, um formulário que aceita palavras-chave para pesquisar livros não precisa estar associado a qualquer classe.

Os campos de um formulário têm as seguintes propriedades (além de outras):

- *Nome*: o nome do campo atribuído pelo designer.
- *Atributo*: se o formulário é associado a uma classe, então o campo pode ser associado a um de seus atributos.
- *Tipo de conteúdo*: define o tipo do campo, que pode ser *string*, *text*, *blob* ou *url*.
- *Pré-carregado*: o campo pode ser pré-carregado com valores que são mostrados ao usuário quando o formulário é renderizado.
- *Modificável*: define se o valor inicial de um campo pode ser trocado ou não. O *default* é que o campo pode ser atualizado.

A Figura 4.9 apresenta um formulário que é associado à classe Livro e sua renderização.

The diagram illustrates a form component with the following structure:

- Visual Representation (Left):** A rounded rectangle labeled "Formulário de Livro". Inside is a small icon of a pencil writing on a checkmark, and below it is the word "Livro".
- Data Structure (Right):** A table with columns for "isbn", "título", "nomeDoAutor", "preço", "númeroDePáginas", "imagemDaCapa", and "quantidadeEmEstoque", each represented by an empty input field.

Figura 4.9. Um formulário e sua renderização.

4.3.7. Componente de Visão *Hierarquia*

Hierarquias são usadas para mostrar dependências entre objetos com dois ou mais níveis. Elas são úteis, por exemplo, para mostrar objetos que se relacionam por uma associação de um para muitos, como *Editora* e *Livro*, ou parte-todo como *Livro* e *Capítulo*. A Figura 4.10 mostra uma definição para uma hierarquia de dois níveis com *Editora* no topo e *Livro* no segundo nível.

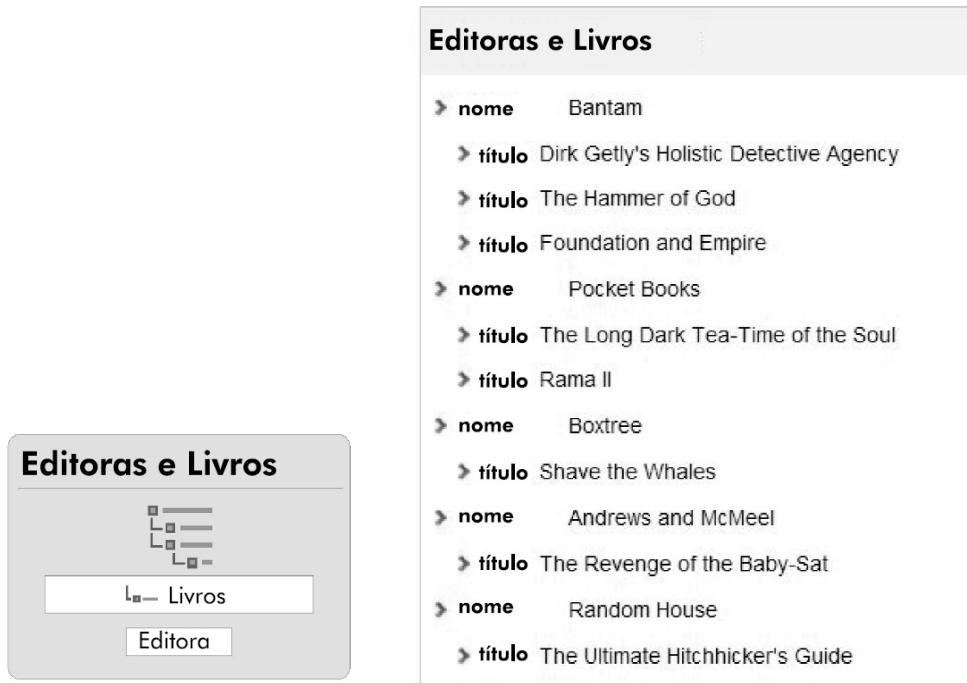


Figura 4.10. Uma hierarquia e sua renderização.

O segundo nível usualmente define uma *expressão condicional baseada em papel*. No exemplo, a expressão condicional baseada em papel define que apenas livros que estiverem ligados a uma dada editora são mostrados abaixo dela. Na ausência dessa

expressão condicional, todos os livros seriam mostrados abaixo de cada editora, mas agora este não é o caso. A renderização da Figura 4.10 mostra uma hierarquia com livros ligados às suas respectivas editoras.

Existem situações que requerem a aplicação do padrão *hierarquia organizacional* (Fowler, 2003) onde um conceito tem subcomponentes que pertencem à mesma classe, a qual pode ser recursivamente aninhada. Para esses casos, IFML fornece o componente de visão chamado *hierarquia recursiva*. A Figura 4.11 mostra uma classe que define uma estrutura organizacional com subestruturas e o componente de visão *hierarquia recursiva* definido para essa classe. Note que a fonte dos dados para essa visão é o nome de papel da associação e não o nome da classe.

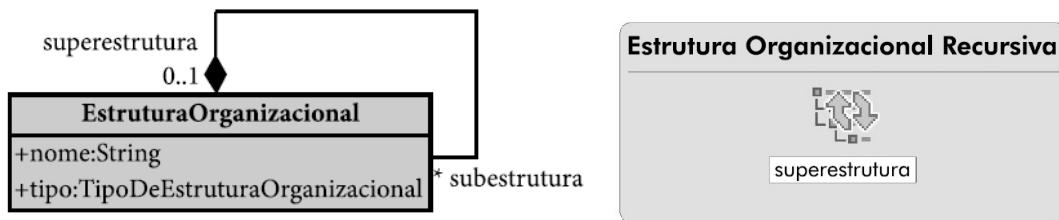


Figura 4.11. Uma classe com associação para si própria e uma visão de hierarquia recursiva para ela.

A definição e a renderização de uma hierarquia recursiva são similares à definição e renderização de uma hierarquia normal. As únicas diferenças são que a classe mestre e a classe mestre são as mesmas e o número de níveis é indeterminado.

4.4. Organização de Hipertexto

Os componentes de visão são os elementos de interface mais locais. Entretanto, é também necessário criar modelos mais amplos para uma interface, incluindo grupos de páginas inter-relacionadas, regiões de navegação, *landmarks*, e assim por diante. Este tipo de modelo é descrito nesta seção e é referenciado aqui como *organização de hipertexto*.

A IFML organiza os componentes de visão em estruturas chamadas *contêineres*. Há basicamente três tipos de contêineres, as páginas, as áreas e as visões de site, as quais são explicadas, nas seções seguintes.

4.4.1. Páginas

Páginas são os elementos de interface que são efetivamente acessados pelo usuário. Páginas são, assim, um tipo de contêiner porque uma página pode conter um ou mais componentes de visão e possivelmente outros contêineres.

A representação IFML de uma página contém um nome e um conjunto opcional de componentes de visão ou subpáginas incluídas. Por exemplo, a Figura 4.12 (origem da referência não encontrada) apresenta uma página que contém uma lista de editoras e uma lista de livros e sua renderização.

O símbolo na Figura 4.12 (origem da referência não encontrada) indica que esta página é uma *homepage*, ou seja, é a página inicial de uma aplicação. O outro símbolo no canto superior direito da página não tem nenhum significado especial em IFML; trata-se apenas de um comando específico da ferramenta para destacar ou obscurecer elementos gráficos.

Os componentes de visão que aparecem dentro da página na Figura 4.12 são independentes um do outro porque não há fluxos (Seção 4.5) entre eles.

Por *default*, componentes de visão definidos na mesma página são renderizados um abaixo do outro e ocupam a largura total da página, mas é possível usar o *modelo de apresentação* como mencionado anteriormente, para definir componentes de visão em diferentes posições e tamanhos. Na ferramenta WebRatio, o modelo de apresentação consiste basicamente em uma grade na qual os componentes de visão são posicionados e dimensionados dentro de uma página. No exemplo da Figura 4.13, origem da referência não encontrada, os componentes de visão são colocados lado a lado.

The screenshot shows a 'Principal' page with two components:

- Lista de Editoras**: A list of publishers with their names and cities. The data is as follows:

nome	município
Andrews and McMeel	Kansas City
Bantam	New York
Boxtree	London
Pocket Books	New York
Random House	New York

- Lista de Livros**: A list of books with their ISBNs, titles, authors, prices, page counts, and quantities. The data is as follows:

isbn	título	nomeDoAutor	preço	número DePáginas	quantidade EmEstoque
0671746723	Dirk Gently's Holistic Detective Agency	Douglas N. Adams	6.99	306	12
0553293370	Foundation and Empire	Isaac Asimov	5.99	282	3
0553286587	Rama II	Arthur C. Clarke and Gentry Lee	6.99	466	2
0752200497	Shave the Whales	Scott Adams	6.99	128	7
055356871X	The Hammer of God	Arthur C. Clarke	6.99	240	1
0671742515	The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99	307	21
0836218663	The Revenge of the Baby-Sat	Bill Waterson	8.95	127	4
0517149257	The Ultimate Hitchhiker's Guide	Douglas N. Adams	12.99	813	6

Figura 4.12. Uma página com dois componentes de visão e sua renderização.

Lista de Editoras		Lista de Livros					
nome	município	isbn	título	nomeDoAutor	preço	número	quantidade
> Andrews and McMeel	Kansas City	> 0671746723	Dirk Gently's Holistic Detective Agency	Douglas N. Adams	6.99	306	12
> Bantam	New York	> 0553203370	Foundation and Empire	Isaac Asimov	5.99	282	3
> Boxtree	London	> 0553206587	Rama II	Arthur C. Clarke and Gentry Lee	6.99	466	2
> Pocket Books	New York	> 0752208497	Shave the Whales	Scott Adams	6.99	126	7
> Random House	New York	> 055350671X	The Hammer of God	Arthur C. Clarke	6.99	240	1
		> 0671742515	The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99	307	21
		> 0836216663	The Revenge of the Baby-Sat	Bill Waterson	8.95	127	4
		> 0517149257	The Ultimate Hitchhiker's Guide	Douglas N. Adams	12.99	813	6

Figura 4.13. Um arranjo de renderização diferenciado para a página mostrada na Figura 4.12
Erro: Origem da referência não encontrada.

4.4.2. Áreas

Áreas são grupos de contêineres que têm afinidades, tais como um conjunto de funcionalidades relacionadas a um objetivo de usuário. Áreas podem conter grupos de páginas relacionadas ou ainda conter outras áreas, caso a interface deva ser estruturada de forma hierárquica. No modelo IFML da Erro: Origem da referência não encontrada
Erro: Origem da referência não encontrada, as áreas aparecem como retângulos sombreados e as páginas como retângulos brancos.

Áreas são por default renderizadas como menus situados no topo das aplicações. As páginas (ou outras áreas) contidas em uma área de nível mais alto são renderizadas como submenus.

4.4.3. Visões de Site

Uma *visão de site* é um pacote que contém uma aplicação inteira que pode ser disponibilizada para usuários. Ela usualmente contém um grupo de *áreas* e *páginas* para a aplicação. A Erro: Origem da referência não encontrada
Erro: Origem da referência não encontrada apresenta um exemplo de uma visão de site com uma página (*Principal*) e duas áreas (*Administração* e *Cliente*).



Figura 4.14. Uma visão de site.

A Erro: Origem da referência não encontrada mostra uma visão de site em estágio inicial de especificação, onde apenas a estrutura de mais alto nível é mostrada. Uma

visão completamente especificada mostraria adicionalmente os componentes de visão dentro das páginas.

Essa estrutura seria renderizada como um menu de alto nível com três opções: *Principal*, *Administrativo* e *Compradores*. O menu *Administrativo* teria como opções *Relatórios* e *Manutenção*. O menu *Compradores* teria como opções *Acesso de Comprador*, *Pedidos* e *Registro*. Já o menu *Principal* não teria opções pois ele acessaria diretamente a página *Principal*.

4.4.4. Home, Landmark e Default

Na Erro: Origem da referência não encontrada Erro: Origem da referência não encontrada, alguns contêineres têm propriedades especiais:

- [H] ou indica a *homepage*. A homepage é a página inicial de uma aplicação. Quando um usuário acessar o *Website*, ele vai acessar a *homepage* por default. Apenas páginas podem ter essa propriedade.
- [D] indica a página *default* de uma área. Quando um usuário navega para uma área específica, ele vai iniciar na página default. Assim, a página default é uma espécie de “*homepage*” para uma área. Apenas páginas podem ter essa propriedade.
- [L] representa uma área ou página *landmark*. Isso significa que a página ou área é diretamente acessível de qualquer outra página dentro da mesma área. Isso evita a necessidade de criar muitos fluxos para páginas que devem ser acessadas de vários lugares.

Apenas uma *homepage* pode existir para uma dada visão de site e apenas uma página *default* para uma dada área. Entretanto, qualquer número de páginas ou áreas *landmark* podem existir em qualquer contêiner.

4.5. Fluxos

Um *fluxo* é uma conexão orientada entre duas páginas ou componentes de visão. Fluxos podem ser usados não apenas para definir possibilidades de navegação entre páginas, mas também para definir dependências de dados. Por exemplo, selecionar um elemento em um componente de visão lista pode fazer com que informação sobre o elemento apareça em um componente de visão detalhes. Isso pode ser conseguido pela definição de um fluxo indo da lista para o componente detalhes.

Existem dois tipos de fluxos em IFML, os fluxos normais de navegação e os fluxos de dados, os quais serão explicados nas subseções 4.5.1 e 4.5.2. Já as subseções 4.5.3 e 4.5.4 descrevem os vínculos de parâmetro simples e multivalorado, que permitem que fluxos normais de navegação e fluxos de dados transportem informação de um componente para outro.

4.5.1. Fluxo Normal de Navegação

Fluxos normais de navegação são um tipo de fluxo que é renderizado como uma ligação clicável em sua página ou componente de visão de origem. Quando uma ligação é clicada, ela provoca a navegação para a respectiva página ou componente de visão. O fluxo é representado por uma seta.

A Figura 4 .15 mostra um exemplo de um fluxo de navegação normal entre duas páginas. A renderização resultante da página *Editoras* contém uma ligação para a

página *Livros* (no canto superior direito). Podemos ver na figura que o fluxo não é ligado aos componentes de visão, mas as páginas. Portanto, a ligação será renderizada apenas uma vez na página na qual a lista de livros é visualizada.

Se o fluxo tem sua origem no componente de visão de lista, como na Figura 4.16, então cada linha na lista deve ter uma ligação individual para a página *Livros*.

Porém, esta lista é ainda apenas um conjunto de ligações que tem o mesmo efeito: navegar para a página *Livros*. As seguintes seções mostram como este tipo de fluxo pode ser usado para transportar informação de um componente de visão para outro, tornando possível, por exemplo, visualizar apenas os livros de uma dada editora.



Figura 4.15. Um uxo de navegação normal entre páginas e a renderização da página de origem.



Figura 4.16. Um uxo de um componente de visão para uma página e sua renderização.

4.5.2. Fluxo de Dados

Fluxos de dados são um tipo de fluxo que não é renderizado, mas que define dependências entre componentes de visão⁷. Setas tracejadas representam fluxos de dados.

Fluxos de dados não definem navegação entre componentes de visão. Eles são usados para obter valores de um componente sem estar navegando a partir dele. Eles são particularmente úteis quando um componente necessita de dados de mais do que um único componente, já que o usuário só poderá estar navegando a partir de um deles.

4.5.3. Vínculo de Parâmetro Simples

Um *vínculo de parâmetro simples* ou simplesmente *vínculo de parâmetro* é uma peça de informação que é transmitida da origem para o destino de um fluxo, seja ele fluxo normal de navegação ou fluxo de dados. Um vínculo de parâmetro tem um *nome* e um *valor*, o qual é obtido no componente de origem.

Um fluxo pode ter vários vínculos de parâmetro, cada um deles com um nome e um valor.

Usualmente, chaves ou valores de atributos de objetos são passados por vínculos de parâmetros. Por *default*, um fluxo de um componente de visão vincula a chave primária do objeto selecionado no componente de origem e a envia ao destino, onde ela pode ser usada, por exemplo, por uma expressão condicional.

Nos exemplos anteriores, apenas constantes foram usadas em expressões condicionais para selecionar os elementos a serem mostrados nos componentes de visão tais como listas e detalhes. Agora, com fluxos e vínculos de parâmetro, é possível usar um componente de visão para definir quais elementos são mostrados em outro componente de visão.

O diagrama IFML da Figura 4.17 mostra uma lista de livros com um fluxo para um componente de visão detalhes. O componente detalhes tem uma condição-chave `[oid=?]`. Como a chave para o livro selecionado na lista é vinculada ao fluxo que vai para o componente detalhes, apenas aquele livro será mostrado quando o componente detalhes for renderizado.

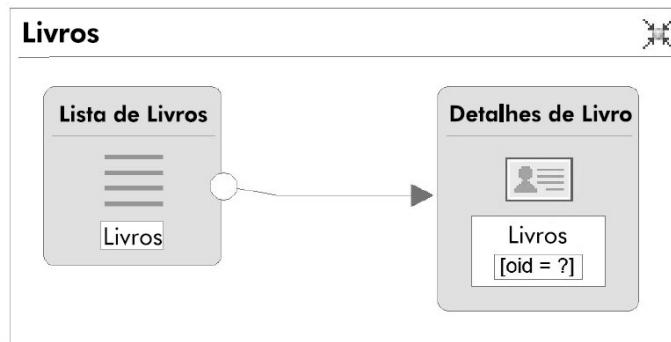


Figura 4.17. Um fluxo de navegação normal conectando dois componentes de visão.

⁷ E operações, como explicado adiante.

Quando o usuário seleciona um livro na lista, ao clicar na respectiva ligação, o componente detalhes apresenta informações sobre o livro. A Figura 4 .18 mostra o estado da página quando a ligação associada ao primeiro livro é clicada.

O vínculo de parâmetro também pode ser usado para associar um formulário a outro componente de visão. Por exemplo, o formulário pode fornecer um campo de pesquisa e uma lista poderia apresentar apenas os elementos que satisfazem o critério de seleção.

A Figura 4 .19 apresenta um exemplo no qual um campo de pesquisa é usado para encontrar os livros de um dado autor. O usuário pode digitar o nome do autor ou parte dele no campo de pesquisa e os resultados são mostrados na lista *Livros*.

O paralelogramo que está próximo ao fluxo indica que este fluxo possui um vínculo de parâmetro que não é o *default*, ou seja, a chave de um objeto selecionado no componente de visão da origem. Embora a ferramenta não mostre o texto associado ao vínculo, ele poderia ser consultado clicando-se no paralelogramo e, possivelmente, seria algo como “*campoX→texto*” indicando que o valor contido no campo *campoX* do componente da origem é passado como o parâmetro cujo nome é *texto* pelo fluxo e pode ser usado no componente de destino em uma expressão condicional como `[nomeDoAutor Contains texto]`, onde *Contains* é um predicado de comparação de cadeias de caracteres que retorna verdadeiro se e somente se a cadeia à esquerda contém integralmente a cadeia à direita.

Na renderização mostrada na Figura 4 .20, o usuário digitou “Clarke” no campo de pesquisa e os resultados produzidos são mostrados na Figura 4 .21.

> Livros

Lista de Livros

título	nomeDoAutor	Detalhes
> Foundation and Empire	Isaac Asimov	Detalhes
> Rama II	Arthur C. Clarke and Gentry Lee	Detalhes
> Shave the Whales	Scott Adams	Detalhes
> The Hammer of God	Arthur C. Clarke	Detalhes
> The Long Dark Tea-Time of the Soul	Douglas N. Adams	Detalhes
> The Revenge of the Baby-Sat	Bill Waterson	Detalhes
> The Ultimate Hitchhicker's Guide	Douglas N. Adams	Detalhes

Detalhes de Livros

isbn	0553293370
título	Foundation and Empire
nomeDoAutor	Isaac Asimov
preço	5.99
númeroDePáginas	282
imagemDaCapa	
quantidadeEmEstoque	7

Figura 4.18. Renderização do modelo da Figura 4.17 quando o primeiro livro da lista é selecionado.

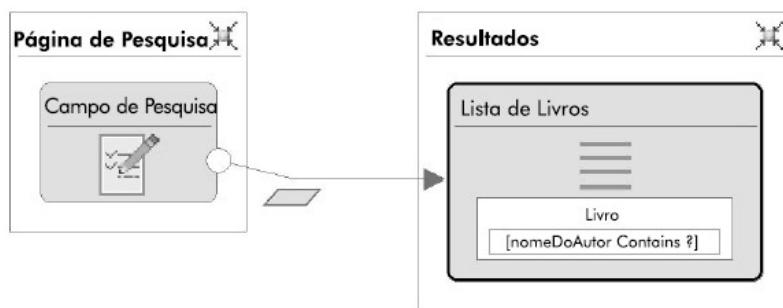


Figura 4.19. Um uxo que transporta dados de um formulário para uma lista.

> Página de Pesquisa

Campo de Pesquisa

Entre com palavra-chave

Figura 4.20. Renderização do modelo da Figura 4.19 antes de iniciar a pesquisa.

> Resultados

Lista de Livros

isbn	título	nomeDoAutor	preço	número DePáginas	imagem DaCapa	quantidade EmEstoque
0553286587	Rama II	Arthur C. Clarke and Gentry Lee	6.99	466		12
055356871X	The Hammer of God	Arthur C. Clarke	6.99	240		2

Figura 4.21. Renderização do modelo da Figura 4.19 após a pesquisa ser realizada.

4.5.4. Vínculo de Parâmetro Multivalorado

É possível definir fluxos a partir de componentes de escolha múltipla tais como listas múltiplas. Nesses casos, um vínculo de parâmetro passa um conjunto de valores (as chaves dos elementos selecionados, por exemplo).

Se uma lista múltipla é ligada a um componente detalhes múltiplos como na Figura 4.22, o usuário pode selecionar vários elementos da lista múltipla como na Figura 4.23 e pressionar o botão “Ver detalhes” para navegar para a página que mostra os detalhes apenas dos livros selecionados (Figura 4.24). O componente detalhes múltiplos tem uma condição-chave que estabelece que apenas livros cujo OID pertença ao conjunto que é vinculado ao fluxo são mostrados.



Figura 4.22. Um uxo para vinculação de parâmetro multivalorada.

> Seleção

Selecionar livros para ver detalhes

título	nomeDoAutor
<input type="checkbox"/> Foundation and Empire	Isaac Asimov
<input checked="" type="checkbox"/> Rama II	Arthur C. Clarke and Gentry Lee
<input type="checkbox"/> The Hammer of God	Arthur C. Clarke
<input type="checkbox"/> The Long Dark Tea-Time of the Soul	Douglas N. Adams
<input checked="" type="checkbox"/> Shave the Whales	Scott Adams
<input checked="" type="checkbox"/> The Revenge of the Baby-Sat	Bill Waterson
<input type="checkbox"/> The Ultimate Hitchhiker's Guide	Douglas N. Adams

[Ver detalhes](#)

Figura 4.23. Renderização do modelo da Figura 4.22 antes de clicar o botão.

> Detalhes

Detalhes Livros

isbn	título	nomeDoAutor	número	preço	DePáginas	imagem	quantidade
> 0553286587	Rama II	Arthur C. Clarke and Gentry Lee	6.99	466		DaCapa	12
> 0752208497	Shave the Whales	Scott Adams	6.99	128			0
> 0836218663	The Revenge of the Baby-Sat	Bill Waterson	8.95	127			21

Figura 4.24. Renderização do modelo da Figura 4.22 após clicar o botão.

4.6. Padrões de Interface Web

Esta seção apresenta alguns padrões de interface considerados bastante comuns em aplicações Web. A maioria dos padrões de interface apresentados nesta seção é adaptada de Ceri *et al.* (2003) e Tidwell (2005). São eles: “índice em cascata”, “índice filtrado”, “tour guiado”, “pontos de vista”, “visão geral mais detalhes” e “navegação em alto nível”.

Não é interesse desta seção esgotar os possíveis padrões, visto que muitos outros existem e ainda podem vir a ser criados, mas mostrar ao leitor a viabilidade de modelar em IFML alguns dos padrões de interface mais comuns, bem como chamar a atenção para a importância de definir padrões de interface de forma a facilitar e padronizar a construção de aplicações.

4.6.1. Índice em Cascata

Um *índice em cascata* é uma sequência de menus baseados em listas que no final atingem um componente detalhes. Por exemplo, um usuário pode iniciar selecionando

uma editora; então ele pode selecionar um livro a partir de uma lista da editora e, finalmente, acessar os detalhes de um livro. A Figura 4.25 ilustra este padrão.



Figura 4.25. Exemplo de um índice em cascata.

No diagrama da Figura 4.25, como os vínculos de parâmetro são *default*, os fluxos carregam os OIDs dos elementos selecionados. Estes OIDs podem ser usados pelas expressões condicionais no componente de visão destino para definir quais elementos devem ser mostrados.

O componente de visão *Lista de Livros* usa uma expressão condicional baseada em papel que mostra apenas os livros ligados à editora cujo OID é vinculado ao fluxo de entrada que vem de *Lista de Editoras*. A expressão condicional em *Lista de Livros* é baseada no papel *EditoraParaLivro*, ou seja, apenas livros ligados à editora na origem do fluxo são mostrados por *Lista de Livros*.

O componente de visão *Livro* mostra a informação completa sobre o livro selecionado em *Lista de Livros*. Os OIDs dos livros selecionados são vinculados ao fluxo que vem de *Lista de Livros* para o componente detalhes. A Figura 4.26 mostra uma sequência de interfaces renderizadas que ilustram este padrão. Em (a), uma editora pode ser selecionada. Em (b), após selecionar a segunda editora, um de seus livros pode ser selecionado. Finalmente, em (c), o terceiro livro foi selecionado e seus detalhes são mostrados.

Uma variação deste padrão consiste em mostrar alguns dados do objeto selecionado em uma das listas intermediárias. A Figura 4.27 ilustra essa situação. Nela, quando uma editora é selecionada, a interface mostra os detalhes da editora e uma lista de livros para seleção. Note que na página intermediária, *Detalhes da Editora* está conectado à *Lista de Livros* por um fluxo de dados. Não é necessário que o usuário navegue de *Detalhes da Editora* para sua lista de livros: ambos os componentes de visão são mostrados ao mesmo tempo quando a página *Selecionar Livro* for acessada. A Figura 4.28 mostra como a página intermediária seria renderizada.

nome	título	nomeDoAutor
Andrews and McMeel	Dirk Gently's Holistic Detective Agency	Douglas N. Adams
Bantam	Foundation and Empire	Isaac Asimov
Boxtree	The Hammer of God	Arthur C. Clarke
Pocket Books		
Random House		

> Detalhes

Livro	
isbn	055356871X
título	The Hammer of God
nomeDoAutor	Arthur C. Clarke
preço	6.99
númeroDePáginas	240
imagemDaCapa	
quantidadeEmEstoque	1

(c)

Figura 4.26. Renderização de *Lista de Editoras*, *Lista de Livros* e *Livro*.



Figura 4.27. Índice cascata com apresentação de dados intermediários.

> Selecionar Livro

Detalhes da Editora		Lista de Livros	
nome	Bantam	título	Douglas N. Adams Detalhes
município	New York	Isaac Asimov Detalhes	
		Arthur C. Clarke Detalhes	
		Foundation and Empire Detalhes	
		Dirk Gettys Holistic Detective Agency Detalhes	
		The Hammer of God Detalhes	

Figura 4.28. Renderização da página intermediária de um índice cascata com dados intermediários.

4.6.2. Índice Filtrado

Em alguns casos, pode ser necessário mostrar uma lista com elementos selecionados. Por exemplo, se a livraria tiver milhares de livros em seu catálogo, não seria viável para um usuário simplesmente procurar por um livro em uma lista. Uma lista filtrada poderia ser usada no lugar disso, por exemplo, pedindo ao usuário para fornecer uma palavra-chave de forma a apresentar uma lista de livros reduzida.

Este padrão é realizado por uma conexão entre um formulário, uma lista e um componente de visão detalhes, como mostrado na Figura 4.29.

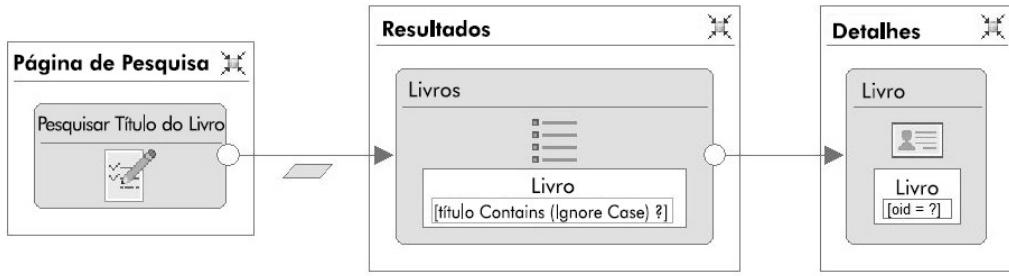


Figura 4.29. Um exemplo de um índice iterado.

Como os resultados são mostrados em uma lista (não uma lista *simple*s), um fator de bloco de *scroll* pode ser definido, bem como a possibilidade de dinamicamente ordenar os resultados. Além disso, o número máximo de resultados pode ser definido para esta lista de forma que listas desnecessariamente grandes não são apresentadas.

4.6.3. Tour Guiado

Um *tour guiado* é um padrão que permite que se visualize os detalhes de um conjunto de objetos, um de cada vez, usando operações de *scroll*. Ele é implementado pelo uso do componente *scroller* com *fator de bloco* 1 ligado a um componente *detalhes*, conforme mostra o lado esquerdo da Figura 4 .30.

O lado direito da Figura 4 .30 mostra uma renderização desta especificação na qual o quinto de oito livros foi selecionado pelo usuário.

4.6.4. Pontos de Vista

Algumas vezes, pode ser interessante apresentar diferentes faces de certos objetos em diferentes momentos. Por exemplo, dados resumidos sobre um livro podem ser apresentados por *default*, mas esses dados podem ser expandidos ou contraídos novamente se o usuário desejar.

Para implementar o padrão de pontos de vista, dois ou mais componentes *detalhes* podem ser definidos para a mesma classe, com diferentes conjuntos de atributos. Fluxos podem ser definidos entre os diferentes componentes para permitir ao usuário mudar de uma visão para outra, como mostra a Figura 4 .31.

O lado esquerdo da Figura 4 .32 mostra a aparência do ponto de vista resumido e o lado direito mostra o ponto de vista da informação completa.

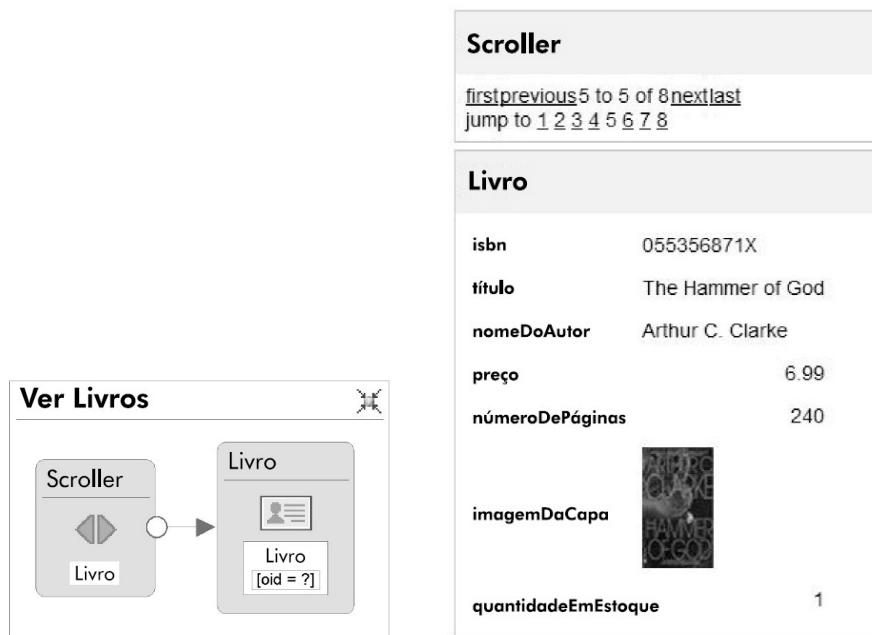


Figura 4.30. Especificação de um *tour* guiado e sua renderização.



Figura 4.31. Especificação de pontos de vista.



Figura 4.32. Dois pontos de vista para um mesmo objeto.

4.6.5. Visão Geral mais Detalhe

O padrão *Visão geral mais detalhe* é muito comum em aplicações tais como visualizadores de e-mail, mas ele pode ser aplicado a uma variedade de outras situações. A ideia é ter na mesma página uma lista com *scroll* de elementos e um componente detalhes de um elemento selecionado em uma região próxima da lista.

A Figura 4.33 mostra uma definição IFML que usa este padrão com uma lista com fator de bloco 3 e um componente detalhes ligado a ela por um fluxo. A Figura 4.34 mostra uma renderização para a página especificada.

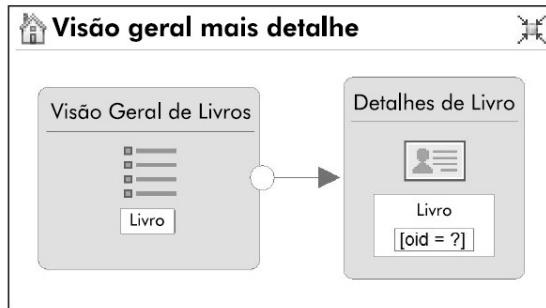


Figura 4.33. Especificação do padrão *visão geral mais detalhe*.

Visão Geral de Livros

first previous 4 to 6 of 8 next last
jump to 1 2 3

título	nomeDoAutor	preço	
► The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99	mostrar detalhes abaixo
► Rama II	Arthur C. Clarke and Gentry Lee	6.99	mostrar detalhes abaixo
► The Hammer of God	Arthur C. Clarke	6.99	mostrar detalhes abaixo

Detalhes de Livro

isbn	0553286587
título	Rama II
nomeDoAutor	Arthur C. Clarke and Gentry Lee
preço	6.99
númeroDePáginas	466
imagemDaCapa	
quantidadeEmEstoque	2

Figura 4.34. Exemplo de renderização de uma página usando o padrão *visão geral mais detalhe*.

4.6.6. Navegação em Alto Nível

Navegação em alto nível é um padrão frequente em páginas Web nas quais um menu de alto nível permite acesso a diferentes áreas do site. O exemplo na Figura 4.35 ilustra navegação em alto nível sendo definida por um conjunto de áreas *landmark*, cada uma contendo uma página (elas poderiam conter mais páginas se necessário).

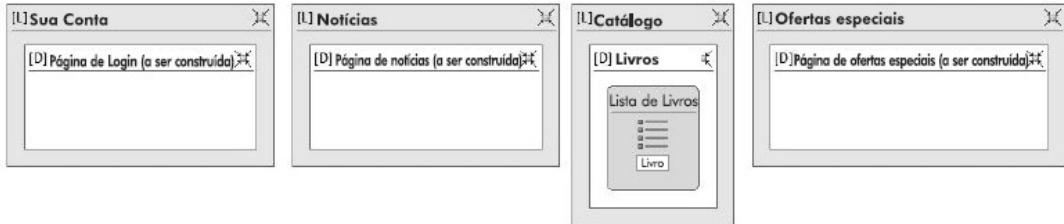


Figura 4.35. Exemplo de especificação de navegação em alto nível.

Para o exemplo, apenas a terceira área foi parcialmente definida. Uma renderização deste modelo é mostrada na Figura 4.36, em que a terceira área, “Catálogo”, é selecionada do menu de alto nível mostrando a página *default Livros*.

Lista de Livros			
título	nomeDoAutor	preço	imagem
> The Ultimate Hitchhiker's Guide	Douglas N. Adams	129	
> Dirk Getly's Holistic Detective Agency	Douglas N. Adams	6.99	
> Shave the Whales	Scott Adams	6.99	

Figura 4.36. Exemplo de renderização de uma página usando navegação de alto nível.

Uma variação desse padrão é a navegação de alto nível baseada em menus. Isso pode ser conseguido pela adição de um conjunto de páginas *landmark* a cada uma das áreas *landmark* de alto nível, como mostrado na Figura 4.37.

A Figura 4.38 mostra a renderização quando o mouse é posicionado sobre o item de menu “Catálogo”.

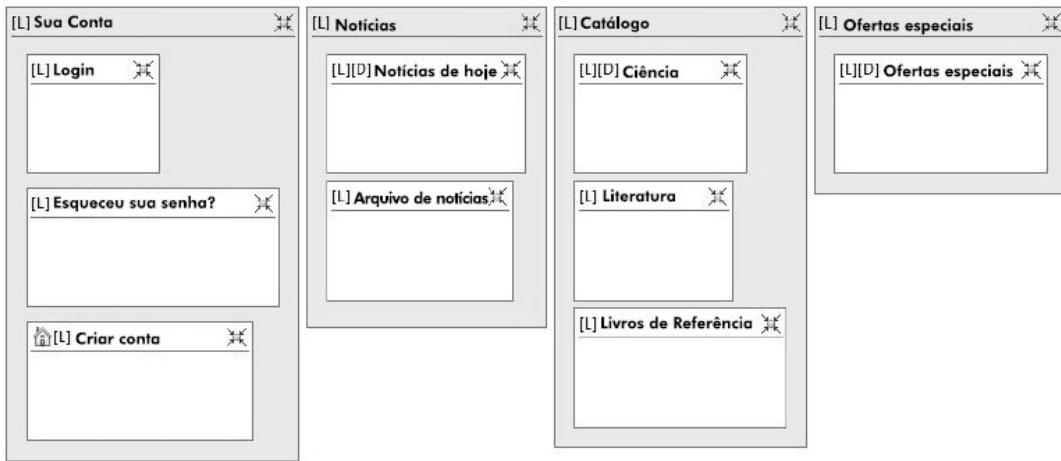


Figura 4.37. Exemplo de especificação de uma navegação baseada em menu de alto nível.

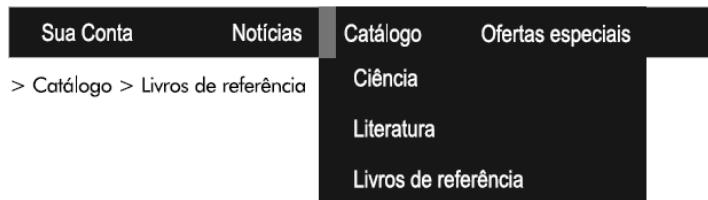


Figura 4.38. Exemplo de renderização de uma página usando navegação baseada em menu de alto nível.

4.7. Modelagem de Operações na Interface

IFML permite a modelagem de operações básicas com o uso de *componentes de operação* que permitem a criação, destruição, atualização, conexão, desconexão e reconexão⁸ de instâncias. Esses componentes realizam as operações mais básicas que podem ser efetuadas sobre objetos. A Figura 4.39 mostra a representação gráfica dessas operações.



Figura 4.39. Componentes de operação: criar, deletar, atualizar, conectar, desconectar e reconectar.

Os componentes de operação não contêm nem renderizam informação: eles apenas a processam. Portanto, eles são representados graficamente fora das páginas.

Cada componente de operação deve ter pelo menos um fluxo de entrada. Algumas vezes, toda a informação que a operação precisa é obtida de um único componente de visão, mas, algumas vezes, mais do que um componente de visão pode ser necessário para fornecer todos os dados necessários para uma operação.

⁸ Substituição de uma ligação.

Componentes de operação têm fluxos de saída especiais que podem ser de dois tipos:

- *Fluxo OK*, que define para onde vai o foco quando a operação foi executada com sucesso sobre todos os objetos.
- *Fluxo KO*, que define para onde vai o foco quando a operação falha para pelo menos um dos objetos afetados por ela.

Os componentes de operação são baseados em classes ou associações, as quais são suas fontes de dados. As operações *criar*, *deletar* e *atualizar* são definidas sobre uma única classe, enquanto *conectar*, *desconectar* e *reconectar* são definidas sobre uma associação e, portanto, têm uma classe *fonte* e uma classe *alvo*.

Por exemplo, uma operação *deletar* poderia ser definida sobre a classe *Livro* como na Figura 4.39; esta operação seria responsável pela destruição de instâncias de *Livro*.

Uma operação *conectar*, no entanto, poderia ser definida sobre uma associação tal como *LivroParaEditora*, como mostrado na Figura 4.39. Portanto, ela seria responsável por conectar (adicionar ligações entre) instâncias da classe *Livro* (fonte) e instâncias da classe *Editora* (alvo).

4.7.1. Operação *Criar*

A operação *criar* cria novas instâncias de uma dada classe. Um fluxo de navegação ou de dados pode ser usado para fornecer dados de inicialização para a nova instância. Usualmente, dados de inicialização são obtidos de um formulário baseado na mesma classe. Um fluxo do formulário para a operação *criar* passa os valores necessários para todos os atributos, exceto pelo OID, o qual é gerado automaticamente pela operação *criar*. Por exemplo, a Figura 4.40 mostra uma estrutura para criar instâncias de *Livro*.

Como podemos ver, o usuário pode digitar no formulário valores de inicialização para um novo livro. O fluxo do formulário vincula dados e leva esses dados para a operação de criação. A operação de criação então cria a nova instância e define seus atributos com os valores recebidos do fluxo.

A Figura 4.41 mostra a renderização do formulário que está na origem do fluxo para a operação *criar*. O fluxo é renderizado como o botão *Salvar*.

Se a operação tiver sucesso, então o fluxo *OK* é seguido para um componente detalhes que mostra o livro que acaba de ser criado. O fluxo *OK* vincula automaticamente o OID do novo livro de forma que a expressão condicional no componente detalhes pode ser usada para definir qual livro apresentar.

Se a operação falhar, o controle vai para uma página de erro, seguindo o fluxo *KO*.

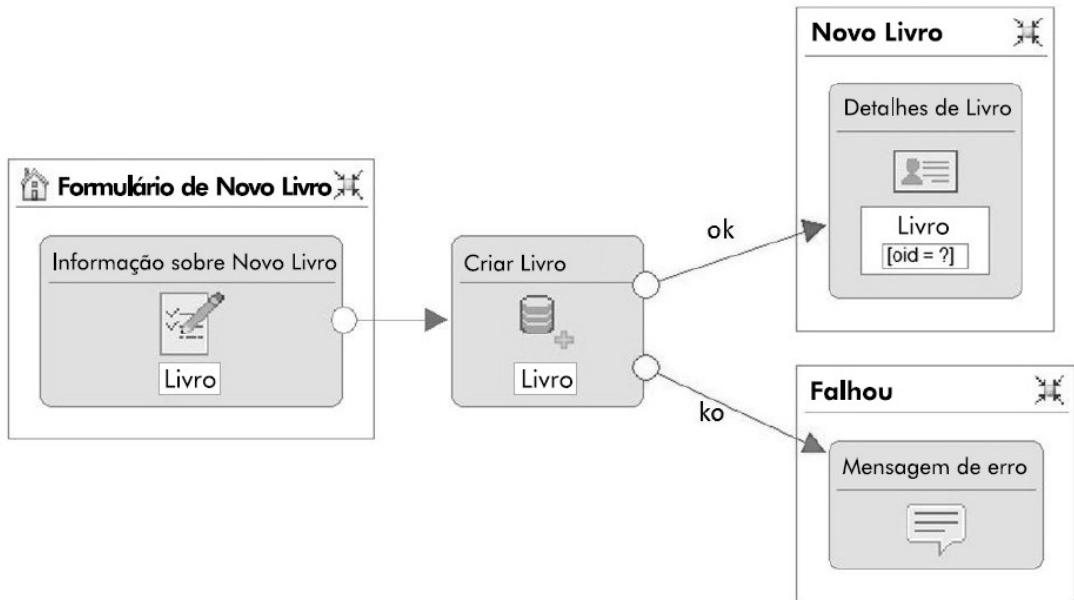


Figura 4.40. Exemplo de aplicação da operação *criar*.

Informação do Novo Livro

isbn	<input type="text"/>
título	<input type="text"/>
nomeDoAutor	<input type="text"/>
preço	<input type="text"/>
númeroDePáginas	<input type="text"/>
imagemDaCapa	<input type="button" value="Escolher arquivo"/> Nenhum Arquivo Selecionado
quantidadeEmEstoque	<input type="text"/>
<input type="button" value="Salvar"/>	

Figura 4.41. Renderização de um formulário com um uxo para uma operação *criar*.

4.7.2. Operação *Deletar*

A operação *deletar* pode ser realizada sobre um ou mais objetos que satisfazem a expressão condicional que a define.

A operação *deletar* pode ser usada em conjunto com uma lista (qualquer tipo de lista), na qual o usuário pode selecionar os objetos a serem deletados. A Figura 4.42 mostra um exemplo no qual o usuário seleciona um livro para deletar de uma lista simples. Note que o seletor da operação *deletar* é baseado no OID do livro, o qual é recebido como um parâmetro do fluxo de entrada.

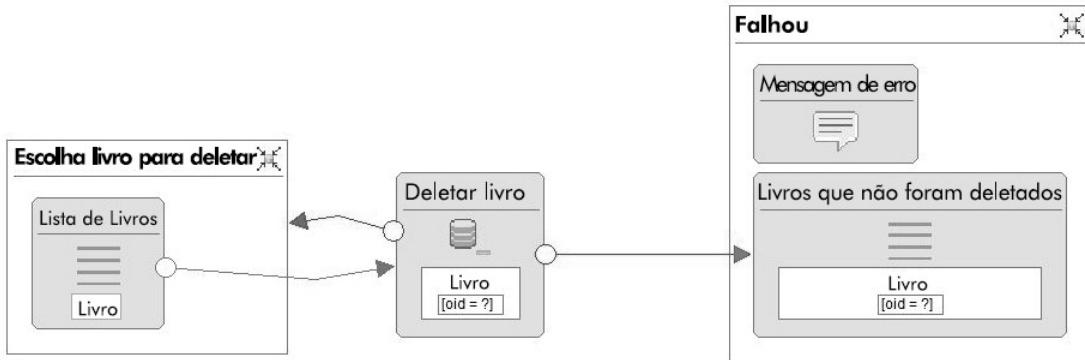


Figura 4.42. Uma operação *deletar* sendo ativada a partir de uma lista.

Esta página pode ser renderizada como na Figura 4.43, na qual a palavra “*deletar*” corresponde ao nome do fluxo.

> Escolha livro para deletar

Lista de Livros		
título	nomeDoAutor	
» Dirk Gently's Holistic Detective Agency	Douglas N. Adams	deletar
» Foundation and Empire	Isaac Asimov	deletar
» Rama II	Arthur C. Clarke and Gentry Lee	deletar
» Shave the Whales	Scott Adams	deletar
» The Hammer of God	Arthur C. Clarke	deletar
» The Long Dark Tea-Time of the Soul	Douglas N. Adams	deletar
» The Revenge of the Baby-Sat	Bill Waterson	deletar
» The Ultimate Hitchhiker's Guide	Douglas N. Adams	deletar

Figura 4.43. Renderização de uma lista simples associada a uma operação *deletar* por um uxo chamado “*deletar*”.

Quando todos os objetos que se qualificam para remoção são realmente removidos, o fluxo OK é seguido. Quando pelo menos um objeto não puder ser deletado, o fluxo KO é seguido e passa como parâmetro vinculado os OIDs dos objetos que não foram deletados. Assim, o fluxo KO poderia, por exemplo, levar a uma lista ou um componente detalhes múltiplos no qual os objetos que não puderam ser removidos são mostrados, como visto na Figura 4.42.

4.7.3. Operação *Atualizar*

Uma operação *atualizar* contém um seletor para um ou mais objetos da mesma classe e um conjunto de atribuições para atualizar os atributos destes objetos. Usualmente, novos valores para atributos são recebidos por fluxos de navegação normais ou fluxos de dados.

A Figura 4.44 mostra como usar uma operação *atualizar* para atualizar o preço de um livro existente.

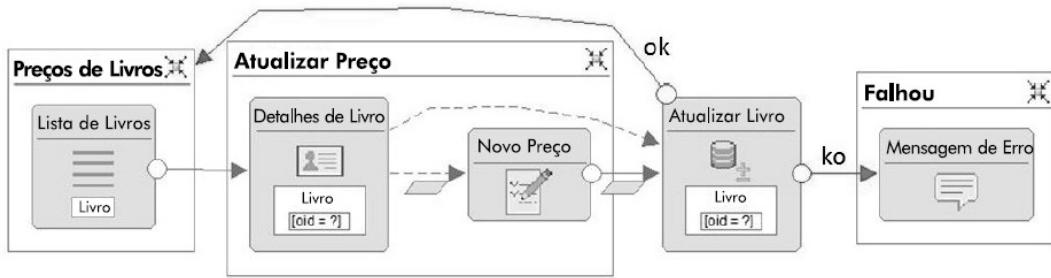


Figura 4.44. Uma operação *atualizar* sendo usada para trocar o preço de um livro.

Na página *Preços de Livros*, um livro é selecionado de uma lista (Figura 4.45). Então uma nova página (*Atualizar Preço*) mostra detalhes do livro e um campo pré-carregado com o preço antigo. O usuário pode mudar este valor e atualizá-lo pressionando “Salvar” (Figura 4.46).

> Preços de Livros

Lista de Livros			
	título	preço	
>	The Ultimate Hitchhiker's Guide	Douglas N. Adams	129 atualizar preço
>	Dirk Getly's Holistic Detective Agency	Douglas N. Adams	6.99 atualizar preço
>	Shave the Whales	Scott Adams	6.99 atualizar preço
>	The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99 atualizar preço
>	Rama II	Arthur C. Clarke and Gentry Lee	6.99 atualizar preço
>	The Hammer of God	Arthur C. Clarke	6.99 atualizar preço
>	Foundation and Empire	Isaac Asimov	5.99 atualizar preço
>	The Revenge of the Baby-Sat	Bill Waterson	8.95 atualizar preço

Figura 4.45. Renderização da página *Preços de Livros*.

O antigo preço é carregado no campo *Novo Preço* por um fluxo de dados de *Detalhes de Livro*. A operação *atualizar* é ativada pelo fluxo de navegação que é renderizado como o botão *Salvar* na página *Atualizar Preço*. Este fluxo de navegação tem um parâmetro vinculado que passa o novo preço do livro do respectivo campo no formulário *Novo Preço* para a operação *atualizar*. Outros atributos do livro, incluindo seu OID, são recebidos por um fluxo de dados de *Detalhes de Livro*.

Na Figura 4.45, o fluxo OK da operação *atualizar* leva de volta para a lista na página *Preços de Livros*, na qual o novo preço pode ser visualizado. O fluxo KO leva para uma janela de erro. Novamente, o fluxo OK é seguido quando todos os objetos qualificados pela expressão condicional são atualizados. Se pelo menos um objeto não puder ser atualizado, o fluxo KO é seguido com o conjunto de OIDs dos objetos que não puderam ser atualizados como parâmetros vinculados.

> Atualizar Preço

Detalhes de Livro	
título	The Revenge of the Baby-Sat
nomeDoAutor	Bill Waterson

Novo Preço	
Novo Preço	<input type="text" value="8.95"/>
<input type="button" value="Salvar"/>	

Figura 4.46. Renderização da página *atualizar preço*.

4.7.4. Operações *Conectar*, *Desconectar* e *Reconectar*

As operações *conectar*, *desconectar* e *reconectar* têm como fonte de dados uma *associação*. As operações *conectar* e *desconectar* têm duas expressões condicionais: uma para o papel de origem e outra para o papel-alvo da associação. A operação *reconectar* pode ser entendida como uma combinação das anteriores. Ela tem três expressões condicionais: uma para o papel de origem, outra para os objetos no papel-alvo que vão ser desconectados e uma terceira para os objetos no papel-alvo que vão ser conectados aos objetos da origem.

As operações *conectar*, *desconectar* e *reconectar* podem adicionar, remover e substituir ligações entre conjuntos de objetos. Se, por exemplo, cinco objetos satisfazem a expressão condicional da origem de um *conectar* e três objetos satisfazem a expressão condicional de destino, então uma operação de conexão iria criar 15 ligações: cada um dos cinco objetos na origem seria ligado a cada um dos três objetos no destino.

O exemplo da Figura 4.47 mostra como uma ligação entre um livro e uma editora pode ser criada. Primeiramente, a ligação é obrigatória do livro para a editora. Assim, não é possível simplesmente pegar uma lista de livros e uma lista de editoras e escolher quais serão ligados. Uma ligação entre um livro e uma editora deve ser criada assim que o livro for criado. Além disso, essa ligação é imutável e não poderá mudar mais tarde.

Na primeira página, *Escolher Editora*, existe uma lista simples de editoras. O usuário deve escolher uma e ser levado para a *Página da Editora*. Lá, os detalhes da editora e uma lista de livros já ligados a ela são mostrados. Note que *Detalhes da Editora* tem uma condição-chave e um fluxo a partir de *Lista de Editoras* e, portanto, ela mostra a editora selecionada na lista. A lista *Livros da Editora* tem um fluxo de dados que vem de *Detalhes da Editora* e uma expressão condicional baseada em papel, de forma que apenas livros que são ligados à editora de *Detalhes da Editora* são mostrados. A Figura 4.48 mostra a renderização de *Página da Editora* após a editora “Bantam” ser selecionada da lista.

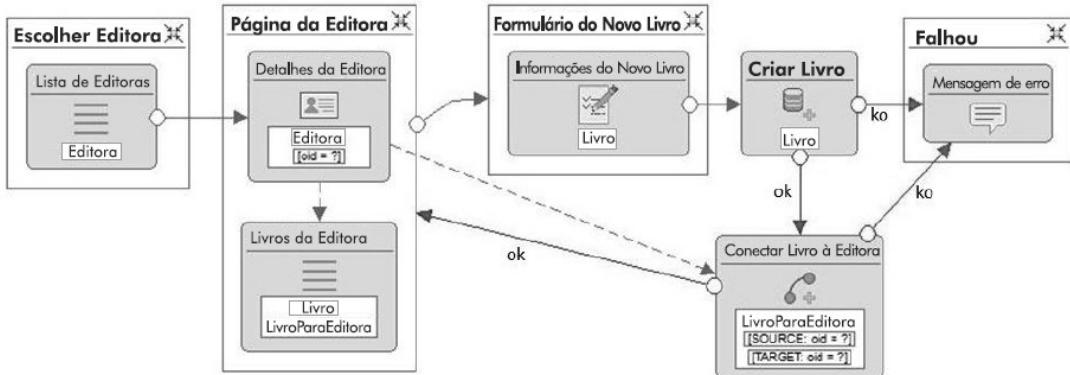


Figura 4.47. Modelo IFML mostrando uma interface na qual livros podem ser criados e ligados a editoras.

A interface "Página da Editora" é dividida em seções:

- Detalhes da Editora**:

nome	Bantam
município	New York
- Livros da Editora**:

título	nomeDoAutor
Dirk Getty's Holistic Detective Agency	Douglas N. Adams
Foundation and Empire	Isaac Asimov
The Hammer of God	Arthur C. Clarke

Figura 4.48. Renderização da Página da Editora.

A partir da *Página da Editora*, o usuário pode navegar para o *Formulário de Novo Livro* usando um fluxo de navegação normal entre ambas as páginas, o qual é renderizado como a ligação *Novo Livro*. Nessa página, o usuário pode preencher o formulário com informações sobre o novo livro, o qual será automaticamente ligado à editora apresentada na *Página da Editora*. Inicialmente, a operação *criar* cria uma nova instância de livro. Seu fluxo OK leva para uma operação *conectar* que é baseada na associação *LivroParaEditora*. O fluxo que sai da operação *criar* tem a chave do novo livro como parâmetro vinculado; esta chave é usada na expressão condicional do papel fonte. A editora é o papel-alvo e ela é obtida pelo fluxo de dados que vem de *Detalhes da Editora*.

Se a operação de conexão for um sucesso, ela retorna o foco para a *Página da Editora*, na qual o novo livro pode ser visto na lista de livros da editora corrente. Se a operação *criar* ou *conectar* falhar, então o foco vai para uma página com uma mensagem de erro.

Entretanto, o design da Figura 4.47 não é seguro. Se um livro for criado e a operação de conexão falhar, este livro ficará inconsistente em relação à sua definição. Para solucionar este problema, IFML permite que uma *transação* seja definida como um grupo de operações. A Figura 4.49 mostra um *grupo de operações* sendo definido para incluir tanto a operação *criar* quanto a operação *conectar*. Como o grupo de operações

pode ser definido como uma transação ($[T]$), então ou todas as operações dentro do grupo obtêm sucesso ou o grupo todo falha. No caso, um único fluxo KO saindo do grupo seria seguido.

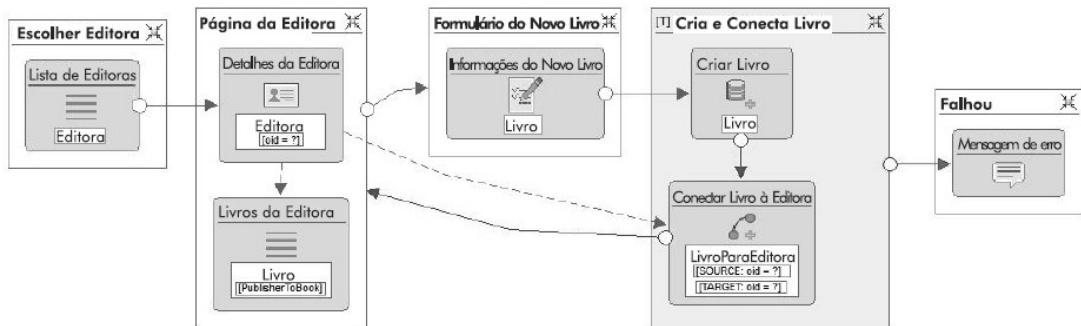


Figura 4.49. Um design mais seguro para o modelo da Figura 4.47 usando um grupo de operações.

4.8. Modelo IFML para operações CRUD

Esta seção mostra passo a passo como implementar uma interface no estilo CRUD para gerenciar informação sobre objetos da classe *Editora*, usando componentes de visão e operações IFML.

A página principal do CRUD *Gerenciar Editoras* apresenta uma lista de editoras existentes. A criação de uma nova editora é iniciada por um clique em uma ligação na página principal que dá acesso a uma nova página com um formulário. Se a criação de uma nova editora tiver sucesso, o controle retorna para a página principal, e a nova editora aparece na lista. Se houver um erro, o controle vai para uma página de erro, a partir da qual é possível retornar à página principal (Figura 4.50).

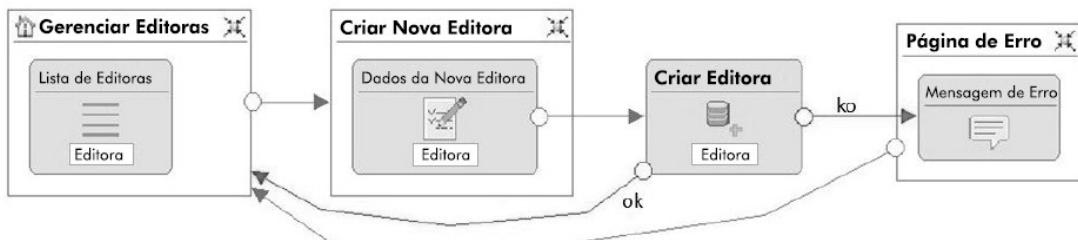


Figura 4.50. De nição de uma interface para *criar*.

A segunda operação de um CRUD, *consultar*, permite a visualização de todos os atributos de uma dada editora. Esta consulta pode ser modelada por um fluxo da lista principal levando para uma página com um componente detalhes, como mostra a Figura 4.51.

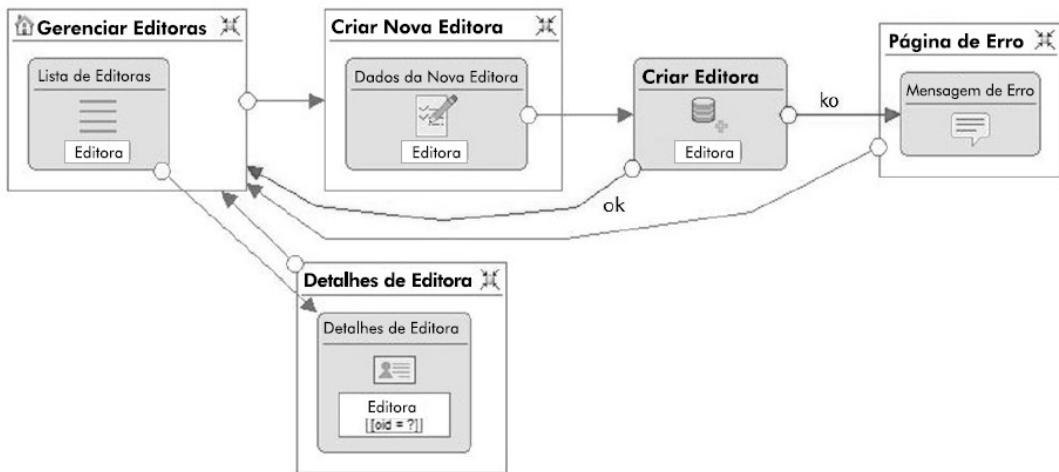


Figura 4.51. Criar e consultar de nidos.

A terceira operação, *atualizar*, permite que uma editora seja selecionada na lista principal e os atributos da editora selecionada são usados para preencher os campos de um formulário com *Editora* como classe fonte. Ali eles podem ser editados pelo usuário e salvos, conforme mostrado na Figura 4.52.

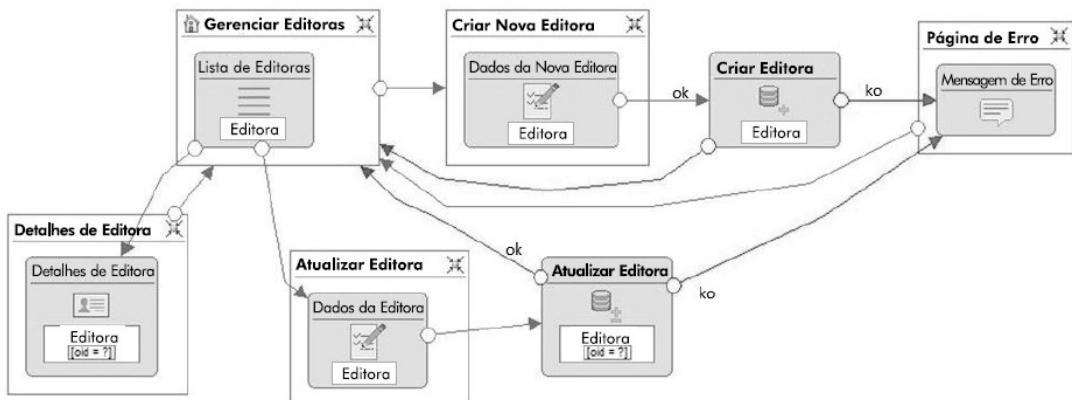


Figura 4.52. Criar, consultar e atualizar de nidos.

Finalmente, a operação para *deletar* uma editora é adicionada, como mostra a Figura 4.53. Uma operação de deleção pode ser acessada a partir da lista principal.

A Figura 4.54 mostra a renderização da página principal do CRUD de *Editora* como definido na Figura 4.53. No topo da página, é possível ver a ligação para a criação de uma nova editora, e logo a seguir está a lista principal, na qual as opções para consulta, atualização e deleção de cada editora estão disponíveis.

A Figura 4.55 mostra a renderização para a página de consulta, a qual é acessada pela seleção da ligação *detalhes* para a terceira editora na lista da página principal. A ligação *retornar* no topo da página permite a um usuário voltar para a página principal.

A Figura 4.56 apresenta a página de atualização da editora, a qual é acessada pelo clique na ligação *atualizar* na lista da página principal. Esta página é idêntica à página para criação de uma nova editora, exceto pelo fato de que ela pré-carrega dados em vez de estar em branco e que, como o atributo *nome* é imutável, seu respectivo campo é definido como não modificável.

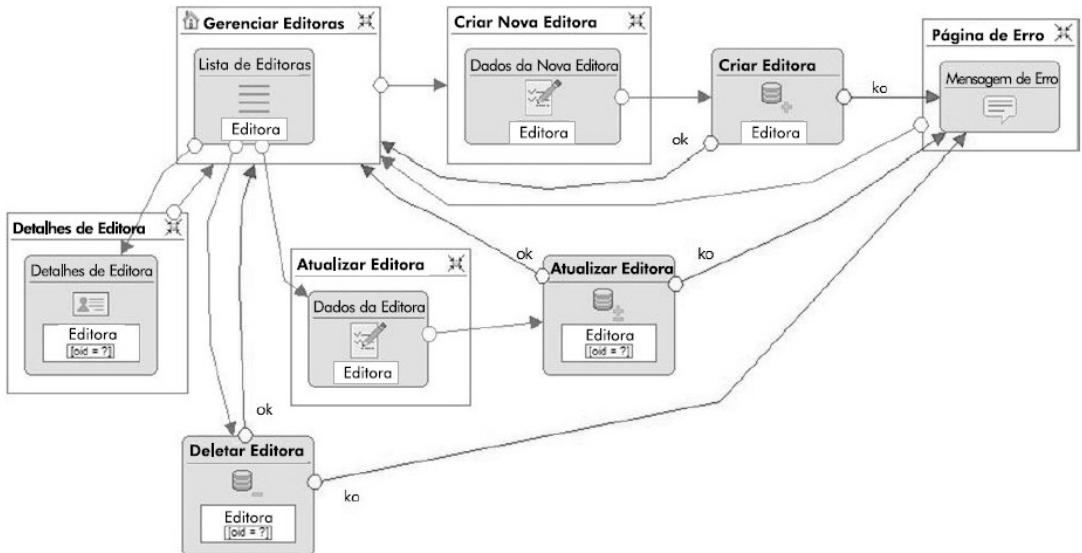


Figura 4.53. Especificação completa de uma interface CRUD.

Criar Nova Editora

> Gerenciar Editoras

Listar Editoras

nome			
Andrews and McMeel	detalhes	atualizar	deletar
Bantam	detalhes	atualizar	deletar
Boxtree	detalhes	atualizar	deletar
Pocket Books	detalhes	atualizar	deletar
Random House	detalhes	atualizar	deletar

Figura 4.54. Janela principal para o CRUD de *Editora*.

Voltar

> Detalhes da Editora

Detalhes da Editora

nome	Boxtree
município	London

Figura 4.55. Renderização para a página de consulta de editora.

> Atualizar editora

Dados da Editora

nome

município

OK

Figura 4.56. Renderização para a página de atualização de editora.

4.9. Modelagem de interfaces de casos de uso com IFML

Conforme visto na seção anterior, trabalhar com as operações básicas neste nível pode rapidamente tornar o diagrama muito complexo para ser facilmente entendido. Uma opção para lidar com essa complexidade é separar o design da interface em diferentes áreas e lidar com uma de cada vez.

Isso será mais eficaz se os *casos de uso de sistema* (Cockburn, 2001) já tiverem sido explorados e a equipe já tiver um entendimento sobre a estrutura do sistema e necessidades do usuário, o que permite que ela faça um design aceitável para a interface.

O design da interface com usuário pode iniciar com um diagrama de caso de uso de sistema como o que é mostrado na Figura 4.57 que corresponde a um pequeno fragmento do diagrama completo, mas que será suficiente aqui para explorar essa questão. O estereótipo `<<report>>` indica que se trata de um caso de uso muito simples que apenas retorna dados aplicando derivações como totalização, ordenação ou filtragem, ou seja, é um caso de uso típico de sistemas de informação usualmente conhecido como *relatório*. Já o estereótipo `<<crud>>` indica que se trata de um caso de uso padrão também típico, no qual uma entidade simples pode ser submetida a quatro operações: criar, alterar, consultar e deletar.

Baseando-se neste diagrama de caso de uso, um possível design de interface que poderia ser preparado teria três áreas principais:

- *Carrinho de compras*: pedidos e pagamento.
- *Minha conta*: login e dados pessoais.
- *Relatórios*.

Este é ainda um design bastante cru, no sentido de que nenhuma questão de usabilidade foi considerada para decidir sobre a melhor organização possível. Entretanto, ele é efetivo. A Figura 4.58 mostra o design IFML dessas áreas.

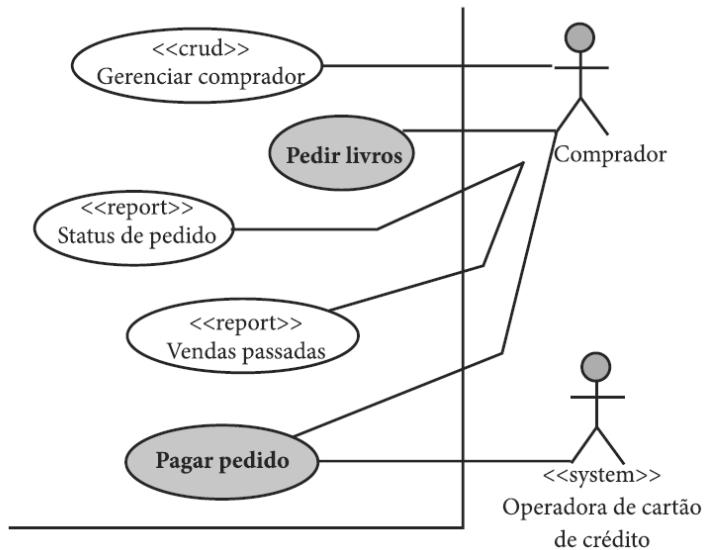


Figura 4.57. Diagrama de casos de uso parcial com casos de uso associados a um ator *Comprador*.

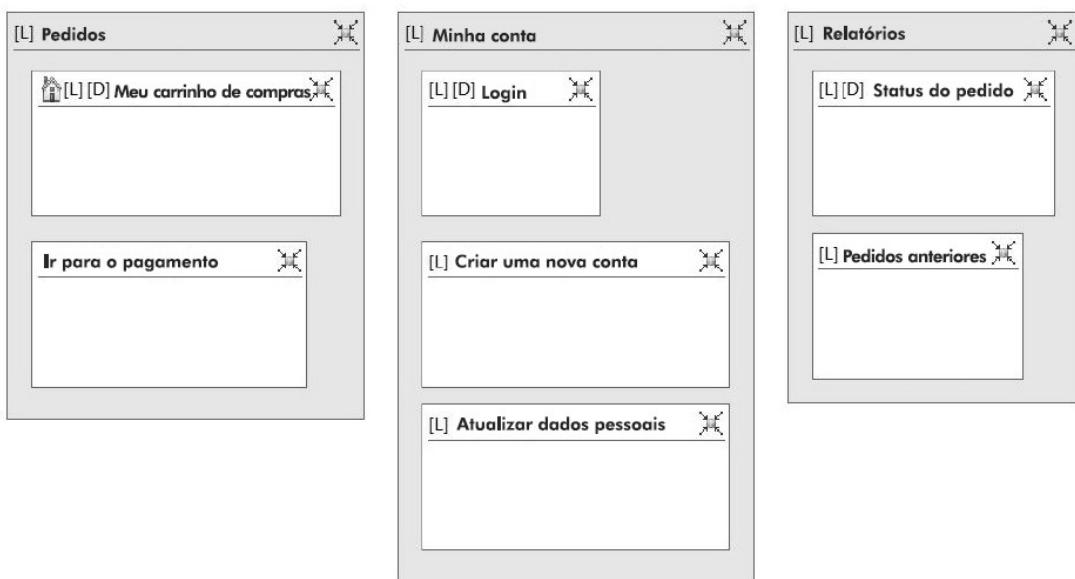


Figura 4.58. Design inicial de alto nível para uma visão de site.

Agora vamos olhar para o caso de uso *Pedir livros* que é mostrado na Figura 4.59.

Caso de uso: *Pedir livros*

- O comprador fornece palavras-chave para pesquisar livros.

2. O sistema gera uma lista de livros para venda que satisfaz as palavras-chave incluindo pelo menos título, autor, preço, número de páginas, editora, ISBN e imagem de capa.
 3. O comprador seleciona livros da lista e indica a quantidade desejada para cada um.
 4. O sistema gera um resumo do pedido (título, autor, quantidade, preço unitário e subtotal para cada livro) e o valor total.
 5. O comprador finaliza o pedido.
-

Exceção 5a: O comprador ainda não se identificou.

5a.1. O comprador fornece uma identificação válida.

5a.2. Retorna ao passo 5.

Variante 5b: O usuário deseja pesquisar mais livros.

5d.1. Retorna ao passo 1.

Figura 4.59. Um caso de uso de sistema.

Para preparar um design de interface baseado nesse caso de uso de sistema, a equipe deve identificar as entradas e saídas de dados (comandos e consultas) necessárias para um ator poder executar o caso de uso. Tanto os comandos quanto as consultas poderão ter parâmetros repassados pelo ator através da interface. As consultas, porém, também irão retornar informações que precisam ser exibidas na interface. Assim, via de regra:

- Operações de sistema, sejam comandos ou consultas, devem ser ativadas em algum lugar da interface. A ativação pode ser obtida pelo clique em um botão ou ligação ou pela seleção de um item em uma lista, entre outras opções. Algumas vezes, o mesmo evento pode ativar mais do que uma operação de sistema: se duas ou mais operações de sistema acontecem imediatamente após um evento de sistema, então elas, possivelmente, são ativadas pela mesma ação do usuário.
- Valores para os parâmetros das operações de sistema precisam ser obtidos de algum lugar. Alguns parâmetros poderiam ser obtidos de outras fontes (tais como um relógio, por exemplo), mas a maioria dos parâmetros seria obtido na interface. Assim, para a maioria dos parâmetros, haverá um componente de visão na interface (usualmente um campo de formulário) que permite que o usuário introduza os dados necessários.
- Resultados obtidos por consultas de sistema devem ser apresentados. Alguns resultados podem estar escondidos do usuário, em alguns casos, mas, usualmente, eles são apresentados. Componentes de visão, tais como listas, detalhes e outros, podem ser usados para mostrar os resultados de consultas de sistema.

Para proceder com o design de interface, as recomendações anteriores são seguidas e os componentes de visão são criados para conter todos os parâmetros e resultados necessários. Ações de operações de sistema (representadas como hexágonos) são adicionadas ao diagrama também. A Figura 4.60 apresenta a evolução do design de interface para a página *Meu carrinho de compras*, a qual implementa parte do caso de uso *Pedir livros*.

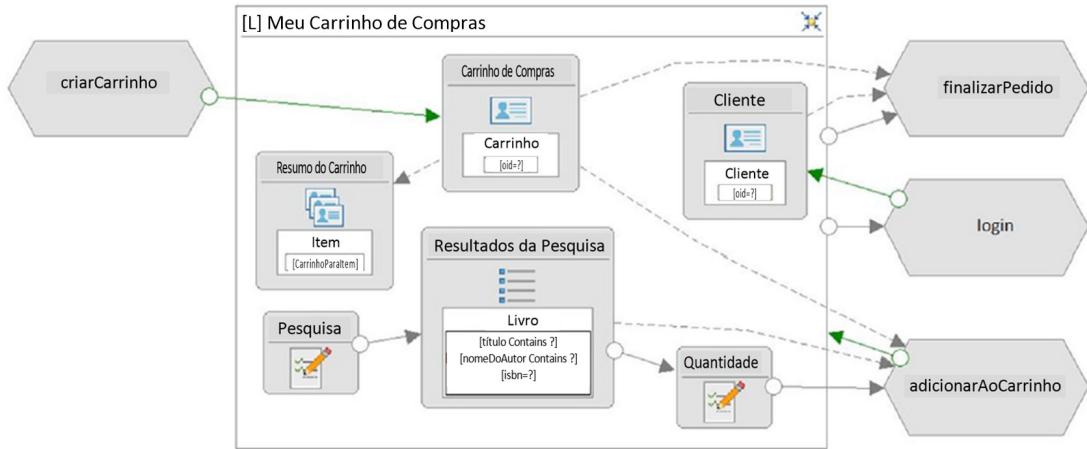


Figura 4.60. Re namento do design de interface para a página *Meu carrinho de compras*.

A operação de sistema *criarCarrinho* produz um novo carrinho que é apresentado pelo componente de visão detalhes *Carrinho de Compras*. O usuário pode usar o formulário *Pesquisa* para pesquisar por livros, os quais são apresentados na lista *Resultados da Pesquisa*. Lá, o usuário pode selecionar um livro e definir a quantidade desejada no formulário *Quantidade*. Isso ativa a operação de sistema *adicionarAoCarrinho*. Essa operação toma parâmetros de *Carrinho de Compras* (o *oid* do carrinho), *Resultados da Pesquisa* (o *isbn*) e *Quantidade* (a *quantidade*). Após um livro ser inserido no carrinho, ele é automaticamente mostrado no componente detalhes múltiplos *Resumo do Carrinho*.

Antes de finalizar o pedido, o comprador deve fazer *login*, caso ainda não o tenha feito. Finalmente, o pedido pode ser finalizado pela execução da operação de sistema *finalizarPedido*, a qual toma parâmetros de *Comprador* (o *cpf* do comprador) e *Carrinho de Compras* (o *oid* do carrinho).

Assim, em vez de usar operações básicas da IFML para definir operações de sistema, a equipe pode usar operações definidas externamente, como na Figura 4 .60. Nesse caso, a equipe pode implementá-los em sua linguagem de programação favorita. Isto é interessante por separar a lógica de transformação dos dados da lógica de interface. As operações definidas externamente podem realizar lógica complexa sobre os dados, mas apenas as entradas e saídas são associadas com os componentes de visão no modelo IFML.

4.10. Considerações Finais

Vimos aqui os conceitos mais básicos e o potencial de modelagem do novo padrão IFML que vem cobrir uma lacuna da linguagem UML por permitir a modelagem do fluxo de interação do usuário com um sistema. Conforme visto, a linguagem de modelagem é gráfica e independente de plataforma. Isto permite uma melhor organização do trabalho de desenvolvimento da parte *front-end* de aplicações computacionais, já que até agora não havia uma notação padronizada para definir tais características, as quais acabam sendo, na maioria das vezes, definidas de maneira informal e implementadas diretamente em código (Object Management Group, 2015).

IFML permite a especificação formal de diferentes perspectivas da aplicação, incluindo, conteúdo de interface, opções de navegação, quais operações são ativadas por quais eventos de interface e até aspectos de sua apresentação.

4.11. Referências

- Brambilla, M., Fraternali, P. (2014). *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufman.
- Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., & Matera, M. (2003). *Designing Data-Intensive Web Applications*. Morgan Kaufmann.
- Chen, Peter (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems* 1(1): 9–36.
- Cockburn, A. (2001). *Writing Effective Use Cases*. Addison-Wesley.
- Eckerson, W. W. (1995). Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems* 10, January.
- Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of reusable object-oriented software. Addison-Wesley.
- Ireland, C., Keynes, M. Bowers, D., Newton, M., Waugh, K. (2009). A Classification of Object-Relational Impedance Mismatch. *Advances in Databases, Knowledge, and Data Applications, DBKDA '09*. p. 36-43. Gosier.
- Miles, R., & Hamilton, K. (2006). *Learning UML 2.0*. O'Reilly.
- Object Management Group (2010). *OCL 2.3.1 Specification*. OMG.
- Object Management Group (2015). *Interaction Flow Modeling Language – Version 1.0*. OMG. Disponível em: <http://www.omg.org/spec/IFML/1.0/>.
- Seffah, A. (2015). *Patterns of HCI Design and HCI Design of Patterns: Bridging HCI Design and Model-Driven Software Engineering*. Springer-Verlag.
- Tidwell, J. (2005). *Designing Interfaces: Patterns for effective interaction design*. O'Reilly Media.
- Wazlawick, R. S. (2015). *Análise e Design Orientados a Objetos para Sistemas de Informação: Modelagem com UML, OCL e IFML*. 3^a edição. Elsevier. (1^a edição em 2004 com o título “Análise e Projeto de Sistemas de Informação Orientados a Objetos”)

Chapter

5

Uma introdução à complexidade parametrizada

Vinicius Fernandes dos Santos, Uéverton dos Santos Souza

Abstract

The solution to many real problems often requires an algorithmic approach, with the subsequent implementation in some system. Typically both the data volume and the frequency of accesses to the system are great, and hence it is necessary an efficient algorithm (traditionally of polynomial time). The theory of NP-completeness was developed to determine which problems probably can not be solved by polynomial algorithms. However, as many NP-hard problems need to be solved in practice, one possibility is to resort to an approximate or heuristic algorithm instead of an exact algorithm. A recent and promising alternative for the treatability of these problems is to use an analysis from the point of view of Parameterized Complexity Theory. This theory, developed by Downey and Fellows, studies the existence of algorithms whose exponential complexity depends only on certain aspects of the input, such algorithms are called fixed parameter tractable (or simply FPT algorithms). This course will introduce the formal concepts of parameterized complexity, basic techniques for designing FPT algorithms, such as bounded search trees and kernelization, and also techniques for negative results, for problems where FPT algorithms are not expected to exist. The examples used will be focused in graph problems.

Resumo

A solução para diversos problemas reais frequentemente exige uma abordagem algorítmica, com a posterior implementação em algum sistema. Tipicamente tanto o volume de dados como a frequência de acessos ao sistema são grandes, sendo portanto necessário algoritmos eficientes (tradicionalmente de tempo polinomial). A teoria da NP-completude foi desenvolvida para determinar quais problemas provavelmente não podem ser resolvidos por algoritmos polinomiais. Entretanto, como muitos problemas NP-difícis precisam ser resolvidos na prática, uma possibilidade é recorrer a um algoritmo aproximativo ou heurístico em vez de um algoritmo exato. Uma recente e promissora alternativa para a tratabilidade desses problemas, é recorrer a uma análise sob o ponto

de vista da Teoria da Complexidade Parametrizada. Esta teoria, desenvolvida por Downey e Fellows, estuda a existência de algoritmos cuja complexidade exponencial depende apenas de certos aspectos da entrada, tais algoritmos são denominados tratáveis por parâmetro fixo (ou simplesmente algoritmos FPT). Este curso, introduzirá os conceitos formais da complexidade parametrizada, técnicas básicas de desenvolvimento de algoritmos FPT, tais como árvores de altura limitada e redução a um núcleo, e também técnicas para resultados negativos, para problemas onde não se espera ser possível desenvolver algoritmos FPT. Os exemplos utilizados serão focados em problema em grafos.

5.1. Introdução

A questão “ $P = NP?$ ” é a mais importante questão em aberto da ciência da computação. E a teoria da NP-completude foi desenvolvida para mostrar que determinados problemas difíceis são, de certa forma, equivalentes. Esta teoria é frequentemente utilizada na identificação de problemas para os quais não se espera a obtenção de algoritmos de tempo polinomial para sua resolução. No entanto, diversos problemas NP-completos e NP-difíceis ainda devem ser solucionados na prática, e portanto é natural perguntar se cada um desses problemas admite algoritmos cuja complexidade de tempo não polinomial seja uma função puramente de alguns aspectos do problema. Questões sobre a existência de tais algoritmos são tratados no âmbito da Teoria da Complexidade Parametrizada desenvolvida por Downey e Fellows [13, 14, 20].

A Teoria da Complexidade Parametrizada surgiu como uma alternativa promissora para se trabalhar com problemas NP-difíceis. Vários problemas de difícil solução podem ser descritos da seguinte forma: “dado um objeto x e um inteiro não negativo k , x tem alguma propriedade que depende de k ?” Na Complexidade Parametrizada, k é chamado de parâmetro. O interesse em tais parâmetros se deve ao fato de que, em muitos casos, somente uma pequena faixa de valores é realmente importante na prática. Logo, o parâmetro k pode ser considerado pequeno em comparação com o tamanho de x . Sendo assim, a intratabilidade (aparente) desses problemas no caso geral pode ser indevidamente pessimista. Desta forma, analisando-se mais profundamente a estrutura da entrada (considerando-a com um conjunto de parâmetros adicionais), deseja-se saber se existem algoritmos determinísticos que são exponenciais somente com relação a k mas polinomiais com relação a x .

Por exemplo, muitas vezes determinar o custo mínimo para a produção de uma empresa, ou o lucro máximo que esta poderá obter, são problemas difíceis de serem resolvidos. No entanto, na prática, as empresas geralmente necessitam determinar apenas se é possível efetuar a produção com um determinado orçamento ou verificar se é possível cumprir uma determinada meta de venda/lucro. Portanto, podemos adicionar aos problemas de custo mínimo ou lucro máximo destas empresas parâmetros fixos adicionais, como um orçamento ou uma meta, dando origem às suas versões parametrizadas, que podem ser mais fáceis de serem solucionadas (admitindo algoritmos de menor complexidade) e ao mesmo tempo satisfazer as necessidades requeridas.

Como podemos observar, a Teoria da Complexidade Parametrizada modela perfeitamente a realidade deste e outros cenários; muitos problemas considerados intratáveis sob o ponto de vista teórico (a menos que $P = NP$), na prática são tratáveis por parâmetro

fixo.

Além disso, um fato notável é que, embora sob o olhar convencional de NP-completude, não haja uma discriminação entre quais problemas NP-completos são mais difíceis ou mais fáceis, na prática, quando deseja-se resolver alguns problemas, é perceptível a diferença de dificuldade em suas resoluções. A complexidade parametrizada, por sua vez, captura de certa forma estas diferenças de dificuldade entre problemas NP-completos.

Uma outra maneira de se observar a importância de algoritmos FPT, decorre do fato de que seu uso fornece uma análise mais refinada da complexidade dos algoritmos. No desenvolvimento e análise de algoritmos, em geral descreve-se a complexidade dos problemas em termos do tamanho da entrada. Entretanto, é comum encontrarmos algoritmos com a mesma complexidade e que frequentemente possuem tempo de execução muito distintos, por dependerem de características específicas da instância que está sendo resolvida, e não apenas de seu tamanho.

A importância dos algoritmos FPT pode ser notada também pelo fato de que diversos algoritmos presentes na literatura, e desenvolvidos antes do surgimento desta teoria, são de fato algoritmos FPT, como o algoritmo de Lenstra para programação inteira [18].

Diversos problemas combinatórios têm sido estudados através da Teoria da Complexidade Parametrizada, como, por exemplo: COBERTURA POR VÉRTICES – provado tratável por parâmetro fixo com um parâmetro k para o tamanho da cobertura [13]; CORTE MÁXIMO – provado ser tratável por parâmetro fixo com um parâmetro k para o tamanho do corte; CLIQUE – provado ser W[1]-completo, onde k é um parâmetro para o tamanho da clique [13]; e CONJUNTO DOMINANTE – provado ser W[2]-completo com um parâmetro k para o tamanho do conjunto [13]. As classes de complexidade W[1] e W[2] são classes de intratabilidade parametrizadas, e serão descritas formalmente mais adiante no texto. Informalmente, estas classes podem ser interpretadas com classes de problemas para os quais não se acredita ser possível o desenvolvimento de algoritmos FPT. Diversos outros resultados podem ser consultados em [13, 14, 20].

Este texto introdutório não se propõe a abordar todos os aspectos da complexidade parametrizada, mas sim fornecer um visão geral da área, através da exposição do leitor aos principais conceitos e com exemplos ilustrativos das principais técnicas. Nossa objetivo é proporcionar um primeiro contato com esta rica e recente área de pesquisa, fornecendo ao leitor interessado referências para aprofundamento posterior.

5.2. Preliminares

Um *problema computacional* é uma questão a ser respondida, tipicamente contendo diversas variáveis cujo valores são não especificados. Uma *instância* de um problema é a especificação de valores para suas variáveis. A descrição de um problema é então dada pela especificação de suas instâncias e a natureza das soluções destas instâncias.

Um *problema de decisão* Π consiste de um conjunto D_Π de instâncias e um conjunto $Y_\Pi \subseteq D_\Pi$ de *instâncias sim*. Um problema de decisão é descrito informalmente pela especificação de: (i) uma instância genérica em termos de suas variáveis; (ii) uma *questão sim-não* declaradas em termos da instância genérica.

Um *problema de otimização* Π consiste de um conjunto D_Π de instâncias, uma função objetivo g e um conjunto S_Π de soluções tais que para cada $I \in D_\Pi$, existe um conjunto associado $S_\Pi[I] \subseteq S_\Pi$ de soluções para I . Um problema de otimização é descrito informalmente especificando: (i) uma instância genérica em termos de suas variáveis; (ii) a função objetivo g a ser calculada, e as propriedades que devem ser satisfeitas por qualquer solução associada a uma instância. Uma solução ótima $S_\Pi[I]$ é a solução que maximiza/minimiza o valor $g(S_\Pi[I])$.

Um *algoritmo* A para um problema Π é uma sequência finita de instruções para um computador, com a finalidade de solucionar Π . Um *algoritmo de tempo polinomial* é definido como um algoritmo cuja sua função de complexidade de tempo é $O(p(n))$, para alguma função polinomial p , onde n é usado para denotar o tamanho da entrada [16].

Definição 1 Um problema Π pertence à classe P se e somente se Π pode ser solucionado em tempo polinomial por um algoritmo determinístico.

Definição 2 Um problema Π pertence à classe NP se e somente se para um dado certificado (uma string que certifica a resposta de uma computação), há um algoritmo determinístico que verifica sua validade em tempo polinomial.

Definição 3 Dados dois problemas Π e Π' , $\Pi \propto \Pi'$ (Π se reduz a Π' em tempo polinomial) se existe um algoritmo que, dado uma instância I de Π , constrói uma instância I' de Π' em tempo polinomial em $|I|$ tal que a partir de uma solução para I' , uma resposta correta para I pode ser emitida em tempo polinomial, e vice-versa.

Definição 4 Um problema Π' é NP-difícil se para todo problema $\Pi \in NP$, $\Pi \propto \Pi'$; se Π' também está em NP, então Π' é NP-completo.

É fácil ver que $\Pi \in P$ implica em $\Pi \in NP$. Se um único problema NP-difícil pode ser solucionado em tempo polinomial, então todo problema em NP pode ser solucionado em tempo polinomial. Se qualquer problema em NP não pode ser solucionado em tempo polinomial, então nenhum outro problema NP-completo poderá ser solucionado em tempo polinomial. Um problema NP-completo Π , portanto, possui a seguinte propriedade: $\Pi \in P$ se e somente se $P = NP$. A questão “ $P = NP?$ ” é a mais importante questão em aberto da ciência da computação.

Um algoritmo é dito *eficiente* se sua complexidade satisfaz algum critério, por exemplo, a complexidade é polinomial no tamanho da entrada. Um problema é dito *tratável* se admite um algoritmo eficiente; caso contrário, o problema é dito ser *intratável*. Como existem diversos critérios que podem ser usados para definir eficiência, há diversos possíveis tipos de tratabilidade e intratabilidade [21].

5.2.1. Tratabilidade Parametrizada na Prática

A teoria da NP-completude foi desenvolvida para mostrar quais problemas provavelmente não admitem algoritmo de tempo polinomial. Desde o início desta teoria em 1971, milhares de problemas têm sido mostrados ser NP-difíceis e NP-completos.

Embora seja interessante saber quais problemas não admitem algoritmos polinomiais, a menos que $P = NP$, um fato inconveniente permanece: esses problemas (especialmente aqueles com aplicações no mundo real) ainda devem ser solucionados. Assim, a seguinte questão emerge:

“Como solucionar um problema NP-difícil da forma mais eficiente possível na prática?”

Primeiramente, temos duas possibilidades:

- ✗ - Tentar construir um algoritmo de tempo polinomial (implica $P = NP$).
- ✓ - Invocar algum tipo de algoritmo heurístico ou aproximativo, de tempo polinomial.

Entretanto, caso estejamos restritos a soluções exatas e ótimas, heurísticas e aproximações podem não ser suficientes. Frequentemente, problemas práticos possuem diversos aspectos adicionais que podem ser considerados como, por exemplo, o grafo de entrada ser planar, as instâncias possuírem grau máximo ou diâmetro constantes, a estrutura buscada possuir tamanho limitado, e assim por diante.

Sendo assim, dado que na prática alguns aspectos do problema possuem tamanho ou valor delimitado, existem outras abordagens para a sua resolução:

- ✗ - Invocar algum tipo de técnica de “força bruta”, que pode ser executada em tempo polinomial com complexidade $O(n^{f(k)})$, onde n é usado para denotar o tamanho da entrada e k é algum aspecto com tamanho ou valor delimitado. Neste caso, quando as instâncias a serem resolvidas são grandes, esta abordagem pode não ser muito viável.
- ✓ - Invocar um algoritmo de tempo não polinomial tal que sua complexidade de tempo não polinomial é *apenas* uma função de algum subconjunto de aspectos do problema que possuem tamanho ou valor delimitado na prática.

Como podemos observar, um problema genérico possui inúmeros aspectos que poderiam ser considerados para análise. No entanto, para cada aplicação existe um subconjunto particular de aspectos do problema que são de fato interessantes. Um *parâmetro*, por sua vez, é uma função que extrai um aspecto ou um conjunto de aspectos particulares de um problema; isto é, um parâmetro é um mecanismo para isolar aspectos de um problema e um “receptáculo” no qual aspectos são encapsulados para manipulação e análise subsequente [21].

Neste ponto, as seguintes questões emergem:

1. Dado um problema e um parâmetro do problema, existe um algoritmo para o problema cuja complexidade de tempo não polinomial é puramente uma função deste parâmetro?
2. Relativo a quais parâmetros do problema tal algoritmo existe?

Se um problema Π para um conjunto K de parâmetros admite um algoritmo como descrito em (1), i.e. executável em tempo $f(K).n^{O(1)}$, então $\Pi \in FPT$ com relação a K (a classe de problemas *tratáveis por parâmetro fixo*). Alternativamente, pode ser possível demonstrar que tal algoritmo provavelmente não existe, estabelecendo uma intratabilidade desta versão do problema. Ao longo do texto, usaremos o termo FPT para nos referirmos tanto à classe dos problemas tratáveis por parâmetro fixo quanto também aos algoritmos que possuem complexidade da forma $f(K).n^{O(1)}$. Assim, podemos dizer que um problema está na classe FPT caso admita um algoritmo FPT.

5.3. Definições

Em contraste com a definição clássica de problema, um *problema parametrizado* consiste de uma primeira componente que pode ser pensada como um problema de decisão ou otimização convencional. Já a segunda componente é o *parâmetro*.

Como podemos observar, na teoria de complexidade parametrizada, estamos interessados em problemas especificados por três itens: (i) a instância de entrada; (ii) o parâmetro a ser analisado; (iii) a questão a ser respondida.

Definição 5 Um problema parametrizado Π é descrito informalmente pela especificação de:

- Uma instância genérica em termos de suas variáveis.
- Os aspectos do problema que constituem os parâmetros.
- Uma questão a ser respondida em termos da instância genérica.

Definição 6 Seja Π um problema NP-difícil e seja $S = \{a_1, a_2, \dots, a_\ell\}$ um subconjunto de aspectos de Π . Denotamos por:

- $\Pi(a_1, a_2, \dots, a_\ell)$, ou $\Pi(S)$, a versão parametrizada de Π onde os aspectos em S são parâmetros fixados.

Definição 7 [13] Um problema parametrizado $\Pi(S)$ pertence à classe XP se existe um algoritmo para solucionar $\Pi(S)$ em tempo $f(S).n^{g(S)}$, onde n é usado para denotar o tamanho da entrada e f e g são funções arbitrárias.

Lema 1 Dado um problema NP-difícil Π e um subconjunto S de seus aspectos, se Π permanece NP-difícil mesmo quando os aspectos em S são delimitados por constantes, então um problema parametrizado $\Pi(S)$ não está em XP, a menos que $P = NP$.

Prova. Se Π está em XP então por definição este problema é solucionado por um algoritmo que pode ser executado em tempo $f(S)n^{g(S)}$ para algumas funções f e g . Quando o valor de todos os aspectos em S são delimitados por uma constante, os valores de $f(S)$ e $g(S)$ são constantes e este tempo de execução torna-se polinomial em n . Como Π é NP-difícil então $P = NP$.

Corolário 2 Se $P \neq NP$, então $\Pi(S)$ pertence a XP se e somente se Π é solucionável em tempo polinomial quando os aspectos em S são delimitados por constantes.

Da mesma maneira que a noção de *tempo polinomial* é central para a formulação clássica de complexidade computacional, a noção central para a complexidade parametrizada é a *tratabilidade por parâmetro fixo*.

Definição 8 Um problema parametrizado $\Pi(S)$ pertence à classe FPT , ou é tratável por parâmetro fixo, se existe um algoritmo para solucionar $\Pi(S)$ em tempo $f(S) \cdot n^c$, onde n denota o tamanho da entrada, c é uma constante arbitrária e f é uma função qualquer.

Dado um problema NP-difícil Π e algum subconjunto de aspectos S de Π , dizemos que S é uma *uma fonte de intratabilidade de tempo polinomial* para Π , se $\Pi(S)$ pertence a FPT .

“ Na teoria da complexidade parametrizada, o foco não está em verificar se um problema é difícil, a teoria parte da suposição que os problemas mais interessantes são intratáveis quando considerados classicamente. O foco desta teoria está na seguinte questão: *O que faz o problema computacionalmente difícil?* ”. [13]

Nas próximas seções exibiremos técnicas para demonstrar que um problema é tratável por parâmetro fixo, e em seguida discutiremos a teoria relacionada à intratabilidade parametrizada de certos problemas.

5.4. Árvores de busca de altura limitada

Diversos algoritmos para problemas combinatórios são recursivos. Estes algoritmos, tipicamente marcam, removem ou rotulam elementos de algum conjunto ou estrutura, como um grafo ou uma *string* a cada chamada recursiva. Normalmente, estes algoritmos vão reduzindo o tamanho da instância até esta ter uma resposta facilmente computável ou até mesmo trivial. Em geral, uma forma simples de se analisar a complexidade de algoritmos desta natureza consiste em calcular a quantidade de chamadas à função recursiva e multiplicar esta quantidade pela complexidade das operações efetuadas dentro de cada chamada.

Pode-se construir uma árvore enraizada para representar as chamadas recursivas: a chamada inicial à função é representada pelo nó raiz e, para cada chamada de uma instância de um procedimento a si mesmo, cria-se um novo nó como filho do nó atual. Esta estrutura das chamadas recursivas pode ser particularmente interessante para se fazer a análise dos algoritmos pois, como pode-se verificar, se todo nó interno da árvore possuir pelo menos 2 filhos, então temos que o número de nós internos é menor que o número de folhas. Assim, para determinarmos um limite para o número total de nós internos, basta calcular o número de folhas, o que é tipicamente mais fácil.

Algoritmos recursivos são particularmente úteis para a resolução de problemas por força bruta. Se, em algum problema, devemos fazer uma escolha dentre k opções,

uma possibilidade é efetuar k chamadas recursivas, com as instâncias modificadas com cada uma destas escolhas. Por exemplo, se desejamos selecionar um subconjunto de vértices de um grafo com determinada estrutura, temos, para cada vértice, duas opções: ele pertence ou não ao conjunto. Esta abordagem é bastante comum na resolução ingênuas de problemas NP-Difíceis e, para instâncias pequenas, podem ser bastante eficientes. Entretanto, o fator de ramificação da árvore, ou seja, o número de filhos de um nó, depende do número de escolhas disponíveis. Se cada nó possui k filhos e a árvore possui n níveis, é fácil verificar que o número de folhas é também, portanto, o número de nós da árvore é da ordem de $O(k^n)$. Se n é o tamanho da instância, este tipo de algoritmo não é um algoritmo FPT. Por outro lado, se tanto o fator de ramificação quanto a altura da árvore são limitados por uma função de um parâmetro, tal algoritmo nos fornece naturalmente um algoritmo FPT, como veremos a seguir. Algoritmos cuja execução corresponde a árvores desta natureza são o assunto desta seção.

Além da forma mais simples apresentada acima, é comum que, em algoritmos mais sofisticados, o número de filhos de um nó e as alturas das folhas não sejam constantes. Em particular, é comum que ramos diferentes da árvore, isto é, subárvores diferentes, possuam tamanhos distintos. Estes fatores, dependem das chamadas recursivas dos algoritmos e de seus parâmetros. Embora de análise mais complexa, diversos algoritmos se utilizam destas assimetrias para a obtenção de tempos de execução mais eficientes. Veremos como utilizar este tipo de artifício nos exemplos a seguir.

Apresentaremos a seguir nosso primeiro algoritmo FPT e, a seguir, verificaremos que, utilizando-se a estrutura do problema, podemos desenvolver algoritmos mais eficientes.

Consideraremos o problema COBERTURA POR VÉRTICES (em inglês, *vertex cover* ou VC). Seja G um grafo e S um subconjunto de $V(G)$. Dizemos que S é uma cobertura por vértices se, para cada aresta e de $E(G)$, pelo menos uma das extremidades de e pertence a S . De forma equivalente, $G \setminus S$ não contém nenhuma aresta. No problema COBERTURA POR VÉRTICES, estamos interessados em encontrar o menor conjunto com esta propriedade, ou seja, uma cobertura por vértices mínima. Em sua versão de decisão, além do grafo G é fornecido também um inteiro k e deseja-se determinar se existe um conjunto de cobertura com no máximo k vértices. A seguir apresentamos a definição formal da versão parametrizada clássica do problema COBERTURA POR VÉRTICES .

Cobertura por Vértices(k)

Instância: Um grafo $G = (V, E)$.

Parâmetro: Um inteiro positivo k .

Questão: G possui um conjunto de vértices I , tal que $|I| \leq k$ e toda aresta de G possui pelo menos um de seus extremos em I ?

O problema COBERTURA POR VÉRTICES é um dos problemas mais clássicos de teoria de grafos e da computação e foi um dos 21 problemas NP-completos estudados no clássico artigo de Karp [17]. Sua importância pode ser verificada pela naturalidade com que problemas reais podem ser modelados como o problema de cobertura. Por exemplo, considere um grafo que modele conflitos, e uma aresta pode representar uma briga entre indivíduos ou tarefas que não podem ser realizadas simultaneamente. Neste caso, COBERTURA POR VÉRTICES consiste em determinar o menor conjunto de vértices cuja remoção

elimina todos os conflitos.

5.4.1. Um primeiro algoritmo para o problema COBERTURA POR VÉRTICES

Da definição do problema, sabemos que em qualquer cobertura por vértices S , cada aresta deve possuir pelo menos uma de suas extremidades em S . Por outro lado, ao inserirmos um vértice v em S , sabemos que todas as arestas adjacentes a ele estão satisfeitas e, portanto, elas não são mais importantes para o grafo. Assim, tanto o vértice inserido em S quanto as arestas que o incidem podem ser removidas do grafo, sem perdermos nenhuma informação relevante. Tendo isso em mente, uma ideia de algoritmo consistiria de escolher uma aresta uv arbitrariamente e efetuar chamadas recursivas com as duas escolhas possíveis: a inserção de u ou de v em S . Note que, como o nosso parâmetro é o tamanho da cobertura, ao inserirmos um elemento em S , só podemos selecionar outros $k - 1$ elementos de G para completar a cobertura por vértices. Em outras palavras, se possuímos uma instância (G, k) , faremos chamadas recursivas para $(G - u, k - 1)$ e $(G - v, k - 1)$. Note que se o nosso parâmetro em alguma chamada for igual a zero, e o grafo possuir alguma aresta, então sabemos que não há solução possível. Por outro lado, se em algum momento não houver mais arestas em G , sabemos que os elementos já inseridos em S são suficientes para cobrir as arestas do grafo. Assim, se nossa instância chegar a alguma dessas situações, sabemos que não há mais chamadas a serem feitas. Como, a cada chamada recursiva o número de arestas e o tamanho do parâmetro diminuem, é fácil ver que o algoritmo para. Como ele explora todas as situações possíveis, é também fácil verificar sua corretude.

De forma mais estruturada, o algoritmo pode ser escrito como a seguir.

Algorithm 1: Primeiro algoritmo para cobertura por vértices.

```
1 CoberturaDeVertices ( $G, k$ ):  
2   se  $G$  não possui arestas então  
3     |   retorna SIM  
4   fim  
5   se  $k = 0$  então  
6     |   retorna NÃO  
7   fim  
8   Escolha uma aresta  $uv$  arbitrariamente  
9   se CoberturaDeVertices ( $G - \{u\}, k - 1$ ) retornar SIM então  
10    |   retorna SIM  
11   fim  
12  se CoberturaDeVertices ( $G - \{v\}, k - 1$ ) retornar SIM então  
13    |   retorna SIM  
14  fim  
15 retorna NÃO
```

Note que, no pior caso, o algoritmo efetua duas chamadas recursivas, tantas vezes quanto possível. Em outras palavras, no pior caso podemos assumir que nenhum nó que potencialmente pudesse ter dois filhos não fará chamadas recursivas. Por outro lado,

como o nosso parâmetro sempre diminui de uma unidade a cada nível, nunca teremos uma altura maior que o parâmetro k inicial. Assim, o número de folhas será no máximo 2^k e, portanto, temos um algoritmo de parâmetro fixo para o problema COBERTURA POR VÉRTICES parametrizado pelo tamanho da cobertura.

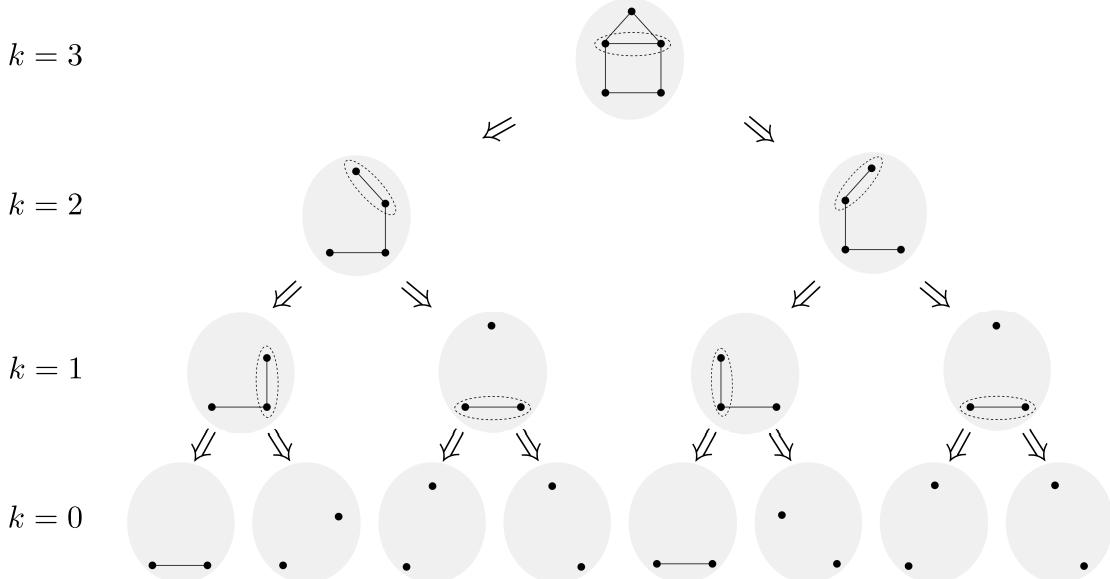


Figure 5.1. Execução do algoritmo para o grafo desenhado no topo, com $k = 3$. Observe que a cada chamada recursiva, duas novas chamadas são feitas, uma para cada extremidade da aresta selecionada. Note que algumas das instâncias do ultimo nível não possuem arestas, portanto em qualquer destes casos o algoritmo retornaria SIM.

5.4.1.1. Utilizando a estrutura do problema para a obtenção de algoritmos mais eficientes

É comum a obtenção de algoritmos mais eficientes através da análise da estrutura dos problemas. Em particular, no caso de grafos, sua estrutura combinatória frequentemente fornece ferramentas para a melhoria dos problemas. Considere o vértice v de maior grau de uma instância de VC. Lembre-se que, o grau de um vértice é o número de arestas incidentes àquele vértice. É fácil verificar que, caso o maior grau, denotado por $\Delta(G)$, seja 1, então o nosso grafo é um conjunto de arestas não adjacentes e possivelmente vértices isolados. Assim, qualquer cobertura por vértices ótima terá exatamente uma extremidade de cada aresta e , portanto, podemos escolher arbitrariamente estas extremidades.

Vamos assumir, então, que o grau de v é pelo menos 2. Note que se v não fizer parte da nossa cobertura por vértice, todos os vértices de $N(v)$, isto é, todos os vizinhos de v , farão parte da cobertura, pois, em caso contrário, alguma aresta estaria descoberta. Assim, em nosso algoritmo, ao invés de remover uma das duas extremidades de uma aresta em cada chamada recursiva, podemos remover um vértice ou todos os seus vizinhos. De fato, é possível verificar que o número máximo de folhas é reduzido neste, caso. Para de-

terminar esta quantidade vamos, primeiramente, definir $T(n)$ como o número máximo de folhas de uma chamada ao nosso procedimento quando o parâmetro é n . É fácil verificar que

$$T(k) = T(k-1) + T(k-d(v)).$$

Por outro lado, como assumimos $d(v) \geq 2$, temos

$$T(k) \leq T(k-1) + T(k-2).$$

Como podemos resolver o problema em tempo constante nos casos em que $k \leq 1$, temos $T(k) = 1$ nestes casos. Mostraremos a seguir que a complexidade deste algoritmo é menor que a versão anterior.

Teorema 3 COBERTURA POR VÉRTICES *pode ser resolvido em tempo $O(n^{O(1)} 1,6181^k)$.*

Prova: Primeiramente, observe que o número de passos em uma chamada recursiva é polinomial em n , portanto, $O(n^c)$ para alguma constante c . Resta mostrar que $T(k) \leq 1,6181^k$. Para isso, usaremos indução. Observe que os casos $k = 0$ e $k = 1$ são claramente verdadeiros. Seja $k \geq 2$. Então, temos $T(k) \leq T(k-1) + T(k-2)$ e, pela hipótese de indução, $T(k) \leq 1,6181^{k-1} + 1,6181^{k-2} \leq 1,6181^{k-2}(1 + 1,6181) \leq 1,6181^k$, onde a última desigualdade é válida pois $1 + 1,6181 \leq 1,6181^2$, o que pode ser facilmente verificado.

5.4.1.2. Explorando ainda mais a estrutura dos grafos

Como acabamos de verificar, é possível utilizar a estrutura dos grafos para desenvolver algoritmos mais eficientes. De fato, exploraremos o raciocínio acima uma vez mais, desenvolvendo um terceiro algoritmo para COBERTURA POR VÉRTICES.

Para obtermos um algoritmo eficiente, utilizaremos o seguinte resultado.

Teorema 4 COBERTURA POR VÉRTICES *pode ser resolvido em tempo polinomial se $\Delta(G) \leq 2$.*

De fato, grafos com grau máximo 2 são a união de ciclos, caminhos e vértices isolados e uma cobertura por vértice para um grafo com esta estrutura pode ser obtida a partir da união das coberturas de vértices das partes que o compõem.

Assim, como no caso acima o algoritmo só fazia chamadas recursivas com vértices de grau pelo menos 2, agora estas chamadas só ocorrerão para vértices com grau pelo menos três. Assim, nossa relação de recorrência para o número de nós também é alterada, passando a ser $T(k) \leq T(k-1) + T(k-3)$ para $k \geq 3$ e $T(k) = 1$, nos demais casos. Usando técnicas similares às anteriores é fácil demonstrar o seguinte resultado:

Teorema 5 COBERTURA POR VÉRTICES *pode ser resolvido em tempo $O(n^{O(1)} 1,4656^k)$.*

Algorithm 2: Segundo algoritmo para cobertura por vértices.

```
1 CoberturaDeVertices ( $G, k$ ):  
2   se  $\Delta(G) < 2$  então  
3     se  $|E(G)| \leq k$  então  
4       retorna SIM  
5     senão  
6       retorna NÃO  
7     fim  
8   fim  
9   se  $k = 0$  então  
10    retorna NÃO  
11  fim  
12 Seja  $v$  um vértice de grau  $\Delta(G)$   
13 se CoberturaDeVertices ( $G - \{v\}, k - 1$ ) retornar SIM então  
14   retorna SIM  
15 fim  
16 se  $\Delta(G) \leq k$  então  
17   se CoberturaDeVertices ( $G - \{N(v)\}, k - \Delta(G)$ ) retornar SIM  
18     então  
19     retorna SIM  
20   fim  
21 fim  
22 retorna NÃO
```

Algorithm 3: Terceiro algoritmo para cobertura por vértices.

```
1 CoberturaDeVertices ( $G, k$ ):  
2   se  $\Delta(G) \leq 2$  então  
3     se Se existe cobertura de tamanho  $\leq k$  então  
4       retorna SIM  
5     senão  
6       retorna NÃO  
7     fim  
8   fim  
9   se  $k = 0$  então  
10    retorna NÃO  
11  fim  
12 Seja  $v$  um vértice de grau  $\Delta(G)$   
13 se CoberturaDeVertices ( $G - \{v\}, k - 1$ ) retornar SIM então  
14   retorna SIM  
15 fim  
16 se  $\Delta(G) \leq k$  então  
17   se CoberturaDeVertices ( $G - \{N(v)\}, k - \Delta(G)$ ) retornar SIM  
18     então  
19     retorna SIM  
20   fim  
21 fim  
22 retorna NÃO
```

Embora estas reduções na base da exponencial possam parecer pequenas, observando a Figura 1.1 é possível verificar que um número reduzido de nós seria gerado pelas chamadas recursivas.

Outra forma de verificar os benefícios da redução da complexidade é através da análise a tabela abaixo, que ilustra o número aproximado de chamadas no pior caso em cada um dos três algoritmos para valores de k iguais a 20, 30 e 40.

Algoritmo	$k = 20$	$k = 30$	$k = 40$
Algoritmo 1	1048576	1073741824	1099511627776
Algoritmo 2	15139	1862776	229199843
Algoritmo 3	2091	95601	4371377

Table 5.1. Número de chamadas à função, em cada uma das versões do algoritmo.

É importante ressaltar que, para valores maiores, estas diferenças se tornam ainda mais discrepantes. Para compreendermos melhor a ordem de grandeza em que estamos trabalhando, considere que cada chamada da função pudesse ser executada em 1 microsegundo. Nos casos em que $k = 40$, teríamos, no melhor algoritmo, um tempo de pouco mais de 4 segundos. Na versão intermediária, este tempo sobe para quase 4 minutos. Já na versão mais ingênua, este tempo chegaria a cerca de 12 dias. Assim, mesmo pequenas melhorias na complexidade podem fazer grande diferença, especialmente por se tratar da base de uma função exponencial.

5.4.2. Cadeia mais próxima

O problema da cadeia (ou *string*) mais próxima pode ser definido da seguinte forma: a partir de um conjunto de cadeias de caracteres de mesmo comprimento, desejamos encontrar uma cadeia que esteja próxima de todas, ou seja, cuja distância para as cadeias de entrada seja a menor possível. A distância entre duas cadeias, neste caso, é medida pelo número de posições nas quais estas cadeias diferem uma da outra. Mais formalmente, podemos definir o problema em sua versão de decisão da seguinte forma:

Cadeia mais próxima

Instância: Um conjunto S de k cadeias de caracteres, de comprimento ℓ e um inteiro d .

Pergunta: Existe uma cadeia s tal que a distância de s para qualquer elemento de S é no máximo d ?

Apresentaremos um algoritmo para este problema parametrizado pela distância máxima d . Em sua versão parametrizada, o problema pode ser definido como a seguir.

Cadeia mais próxima parametrizada pela distância

Instância: Um conjunto S de k cadeias de caracteres, de comprimento ℓ .

Parâmetro: Um inteiro positivo d .

Pergunta: Existe uma cadeia s tal que a distância de s para qualquer elemento de S é no máximo d ?

Antes de fornecer um algoritmo para este problema, apresentaremos uma observação que, embora simples, permite demonstrar que o problema é tratável por parâmetro

fixo.

Observação 1 *Se existem $x, y \in S$ tais que $\text{dist}(x, y) > 2d$, então a resposta é NÃO.*

Note que a condição acima pode ser verificada em tempo polinomial. Assim, caso haja um par de cadeias com distância maior que $2d$, podemos simplesmente responder NÃO. A partir de agora, assumiremos que não existe um par de cadeias com tal propriedade.

Um procedimento recursivo para resolver este problema pode funcionar da seguinte forma: inicialmente escolhemos arbitrariamente uma cadeia s_1 de S , e fazemos $s = s_1$. Caso a distância de s para todos os elementos de S seja no máximo d , não há mais nada o que fazer e podemos responder SIM. Caso contrário, existe uma cadeira $s_i \in S$ tal que $\text{dist}(s, s_i) > d$. Tentaremos, então, modificar posições de s , de forma que a distância em relação a s_i se torne no máximo d . Note que, tomando $d + 1$ posições que tenham caracteres distintos em s e s_i , pelo menos um deles deve ser alterado, caso contrário s não terá distância no máximo d em relação a s_i . Podemos, então, testar todas essas possibilidades, com cada uma dando origem a uma chamada recursiva do procedimento. Para cada uma destas possibilidades, podemos marcar esta posição como permanentemente alterada de forma que não voltemos atrás desta decisão (caso múltiplas alterações fossem feitas na mesma posição, poderíamos efetuar apenas a última). Observe ainda que a cada alteração é efetuado um novo conjunto de chamadas recursivas e, assim, a distância de s aumenta em uma unidade em relação à cadeia inicial s_1 . Como a solução final s deve satisfazer $\text{dist}(s, s_1) \leq d$, podemos ter no máximo d níveis de profundidade, totalizando $O((d+1)^d)$ chamadas recursivas.

Uma análise cuidadosa das ideias apresentadas pode fornecer um algoritmo de complexidade $(d+1)^d |S|$, como pode ser verificado em [12].

5.4.2.1. Determinando a complexidade de algoritmos recursivos

Embora uma abordagem abrangente sobre a determinação da complexidade de algoritmos recursivos esteja fora do escopo deste texto, abordaremos este aspecto de forma breve. De certa forma, as constantes obtidas nos algoritmos para o problema da cobertura por vértices podem parecer obtidas num truque de mágica mas, na verdade, existem técnicas para analisar de maneira satisfatória a complexidade de algoritmos desta natureza.

Algoritmos recursivos têm, normalmente, sua complexidade analisada através do uso de relações de recorrência. Assim, o tempo de execução de uma função é calculado em função do número de passos executados dentro da função e também do tempo de execução das chamadas recursivas, que são representados de forma implícita. Os casos base da relação de recorrência, em geral são obtidos quando os argumentos da função são, de alguma forma, pequenos (em nossos exemplos, quando o parâmetro k era demasiadamente pequeno ou quando a entrada fica simples, como no caso em que o grafo possuía alguma propriedade que o tornava “simples”, como o grau máximo pequeno).

Para a resolução de relações de recorrência, recomendamos uma consulta a um texto clássico de algoritmos, como [8].

5.5. Redução a um núcleo

A forma mais natural de demonstrarmos que um problema Π é FPT , é exibindo um algoritmo de tempo $f(k).n^\alpha$ que solucione Π . Muitas vezes, a construção desse algoritmo não ocorre de forma imediata, sendo necessário o uso de técnicas que auxiliem a sua construção. Dentre as técnicas conhecidas para classificação de um problema como FPT , uma das mais utilizadas é o método de *redução a um núcleo* do problema.

A idéia principal deste método é reduzir, em tempo polinomial, uma instância I do problema Π a uma instância I' , tal que o tamanho de I' seja limitado por alguma função do parâmetro k independente do tamanho de I . Dessa forma, a instância I' constitui um núcleo do problema com tamanho limitado. Logo, I' pode ser exaustivamente analisada, e sua solução pode ser apresentada como uma solução para I , caso exista.

De certa forma, o conceito de redução a um núcleo (em inglês, *kernelization*) pode ser pensado como a formalização da ideia de pré-processamento. Em diversos problemas é comum a possibilidade de redução do tamanho da instância através da eliminação de partes da instância que sejam trivialmente resolvíveis ou que não façam parte de solução alguma.

Como exemplo, consideremos o problema da clique máxima, que consiste em encontrar o maior conjunto de vértices tal que existe aresta entre todos os seus elementos. Assumindo que o grafo contenha alguma aresta, podemos eliminar todos os vértices isolados, pois estes certamente não farão parte da maior clique. Por outro lado, caso haja algum vértice universal, sabemos que ele fará parte de qualquer clique máxima, então podemos removê-lo e diminuir uma unidade da resposta do problema reduzido.

A técnica de redução a um núcleo consiste em efetuar tantos passos de pré-processamento quanto possível de forma que, ao final a solução seja trivial ou então que a instância seja reduzida de forma que seu tamanho seja limitado por um polinômio do parâmetro k original. Esta instância reduzida é chamada de núcleo que pode ser pensada como a parte mais difícil do problema.

Formalmente, se a instância I de um problema consiste do par (x, k) , onde x é a entrada e $k \in \mathbb{N}$ é o parâmetro, o *núcleo* de I é uma instância $I' = (x', k')$ tal que $|x'| \leq g(k)$ e $k' \leq g(k)$, onde g é uma função computável qualquer, que depende apenas de k , tal que I é uma instância SIM se e somente se I' é uma instância SIM. Assim, desejamos obter uma transformação f de forma que $f(I) = I' = (x', k')$, satisfaça $I \in Y_\Pi$ e somente se $I' \in Y_\Pi$. Em particular, estamos interessados em transformações que podem ser feitas em tempo polinomial em $|x|$ e k , ou seja, cujo tempo é um polinômio que depende do tamanho da entrada e do parâmetro k . Chamamos tais reduções de *regras de redução*.

Um algoritmo de redução a um núcleo consiste, portanto, em um conjunto de regras de redução tais que, quando aplicadas à exaustão, levam uma instância $I = (x, k)$ a uma instância final $I' = (x', k')$ tal que $|x'| \leq g(k)$ e $k' \leq g(k)$, onde g é uma função computável qualquer, que depende apenas de k . Em termos mais informais, o algoritmo de redução consiste de sucessivas aplicações, tantas vezes quanto possível, de regras de redução até obtenção de uma instância final reduzida que é limitada pelo parâmetro inicial k .

O papel de um núcleo pode, à primeira vista, parecer secundário na teoria de algoritmos FPT. Entretanto o teorema a seguir nos mostra a equivalência entre a existência de um núcleo e de um algoritmo tratável por parâmetro fixo.

Teorema 6 *Um problema Π admite um algoritmo tratável por parâmetro fixo se e somente se ele admite um núcleo.*

De fato, como veremos na Seção 1.8, esta equivalência permite um nível mais refinado de classificação da dificuldade de problemas FPT. Mas neste momento, nos concentraremos na obtenção de núcleos para alguns problemas.

5.5.1. Um núcleo para o problema de cobertura de vértices

Como visto na seção anterior, o problema de cobertura de vértices admite algoritmos tratáveis por parâmetro fixo. Apresentaremos, agora, um algoritmo de redução a um núcleo para o mesmo problema. Para este fim, definiremos regras de redução e, em seguida, demonstraremos que, após a aplicação das regras, a instância resultante é limitada pelo tamanho do parâmetro k .

Seja (G, k) uma instância do problema de cobertura de vértices parametrizado pelo tamanho da cobertura, onde G é um grafo e k é o parâmetro. Uma primeira observação diz respeito a vértices isolados, ou seja, vértices que não possuem nenhuma aresta incidente a eles. Note que vértices isolados não farão parte de nenhuma cobertura de vértices ótima, uma vez que sua inclusão em uma cobertura não cobriria nenhuma aresta. Além disso, sua remoção do grafo também não afeta nenhuma aresta existente. Assim, podemos definir nossa primeira regra de redução.

Regra de redução 1 *Se (G, k) é uma instância do problema de cobertura de vértices parametrizado e o grafo G contém um vértice isolado v , então remova v de G , obtendo uma nova instância $(G - v, k)$.*

Voltemos, agora, nossa atenção a vértices de grau elevado. Seja v um vértice de grau pelo menos $k + 1$, ou seja, existem pelo menos $k + 1$ arestas incidentes a v . Note que, como todas estas arestas precisam ser cobertas, caso v não faça parte da cobertura de vértices, a única maneira de cobrir as arestas incidentes a ele é incluindo $N(v)$ à cobertura. Entretanto, como o grau de v é pelo menos $k + 1$, precisaríamos adicionar pelo menos $k + 1$ vértices, o que não é possível, pois podemos adicionar no máximo k elementos. Assim, concluímos que v deve fazer parte de qualquer cobertura de vértices de cardinalidade no máximo k . Podemos, então, formular nossa segunda regra de redução.

Regra de redução 2 *Se (G, k) é uma instância do problema de cobertura de vértices parametrizado e o grafo G contém um vértice v com $d(v) \geq k + 1$, então remova v de G e reduza uma unidade de k , obtendo uma nova instância $(G - v, k - 1)$.*

Note que aplicando as regras de redução 1 e 2 tanto quanto possível, a instância resultante possuirá grau máximo k , uma vez que a existência de vértices com grau superior

a k permitiriam a aplicação da Regra de redução 2 mais vezes. Embora estas regras sejam consequências de observações simples, a adição de apenas uma terceira regra, mais técnica, nos permitirá obter o núcleo que buscamos.

Regra de redução 3 *Se $I = (G, k)$ é uma instância do problema de cobertura de vértices parametrizado tal que as Regras de redução 1 e 2 não podem ser mais aplicadas e alguma das condições abaixo é satisfeita, então I é uma instância NÃO.*

- $k < 0$, ou
- G possui mais de $k^2 + k$ vértices, ou
- G possui mais de k^2 arestas.

O caso em que $k < 0$ é de fácil verificação: significa que já adicionamos mais vértices ao conjunto de cobertura do que poderíamos. Para verificar a corretude das demais condições, observe que com k vértices, é possível cobrir no máximo k^2 arestas, uma vez que o grau máximo é k . Para o limite no número de vértices, note que cada vértice tem grau pelo menos 1, uma vez que a Regra de redução 1 não pode ser mais aplicada. Assim, como todo vértice possui grau pelo menos 1 e toda aresta deve ser coberta, então cada vértice deve possuir pelo menos um vizinho em qualquer conjunto de cobertura ou fazer parte do próprio conjunto. Assim, como no conjunto de cobertura existem no máximo k vértices e cada um desses pode ter no máximo k vizinhos fora do conjunto e todos os vértices do grafo devem se encontrar em uma destas situações, temos no máximo um total de $k + k^2$ no grafo.

Podemos então concluir o seguinte resultado.

Teorema 7 *O problema de cobertura de vértices admite um núcleo com no máximo $O(k^2)$ vértices e $O(k^2)$ arestas.*

De fato, após a aplicação das regras, ou teremos concluído que a instância em questão é uma instância NÃO, através da Regra de redução 3, ou teremos uma instância cujo tamanho não poderá ser maior que $O(k^2)$. Um exemplo de aplicação das regras de redução pode ser visto na Figura 1.2

Assim como no caso das árvores de altura limitada, em que pudemos melhorar substancialmente nosso algoritmo inicial utilizando a estrutura do problema, o mesmo ocorre com os algoritmos de redução a um núcleo. Embora fora do escopo deste livro, é possível formular regras de redução de forma a obtermos um núcleo de tamanho linear para o problema [9].

5.5.2. Conjunto de arestas de retroalimentação em torneios

Um *torneio* é um grafo direcionado tal que, entre cada par de vértices u e v , existe exatamente uma aresta direcionada, (u, v) ou (v, u) . Um conjunto de arestas de retroalimentação em um grafo direcionado consiste em um conjunto de arestas que, se removidas do grafo, deixam o grafo acíclico. Um grafo direcionado é acíclico se não é possível, para nenhum

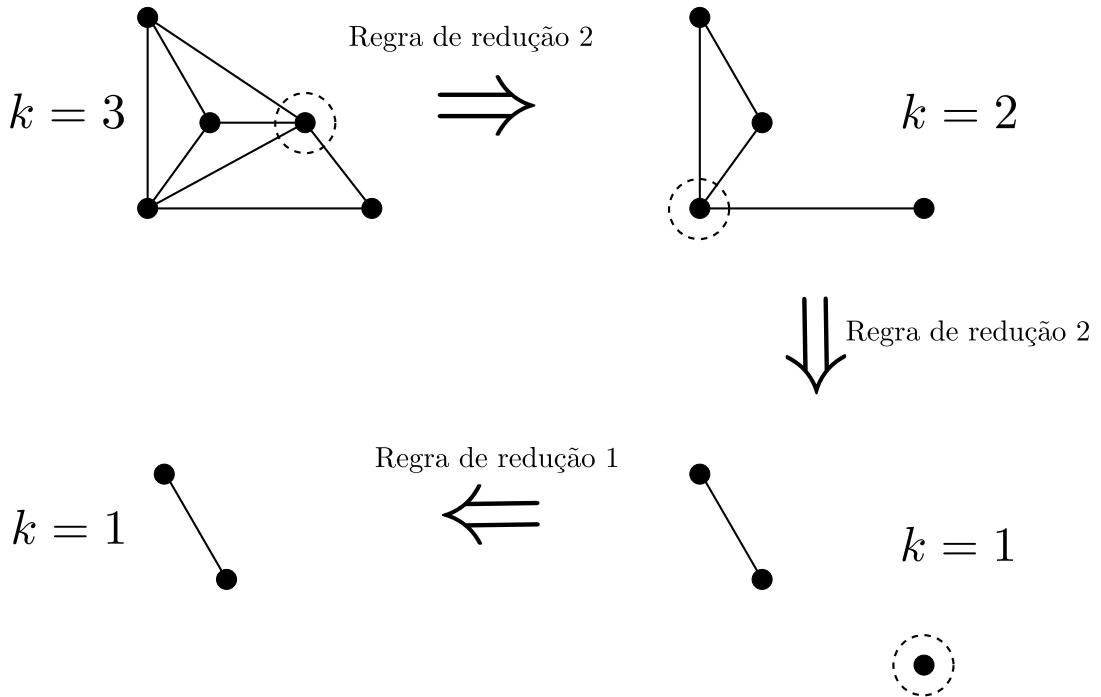


Figure 5.2. Exemplo de aplicação das regras de redução 1 e 2.

vértice inicial v , retornar a ele mesmo, após seguir uma sequência de arestas, respeitando suas direções. Nos restringiremos, aqui, ao problema do conjunto de arestas de retroalimentação no caso de torneios. Consideraremos, em particular, a versão parametrizada pelo tamanho do conjunto de retroalimentação.

Ao nos restringirmos a torneios encontramos uma dificuldade: ao remover uma aresta de um torneio, ele deixa de ser um torneio. Assim, teríamos que encontrar regras de redução que não removessem arestas do grafo original. Para contornar esta limitação técnica, exploraremos as relações entre conjuntos de arestas de retroalimentação e a reversão de arestas, ou seja, a operação que transforma uma aresta (u, v) em uma aresta (v, u) . O seguinte fato pode ser verificado facilmente.

Observação 2 Se G é um grafo direcionado e F é um subconjunto de arestas de G tal que a reversão de F deixa o grafo acíclico, então F é um conjunto de retroalimentação.

É possível verificar que a recíproca não é verdadeira. Entretanto, alterando ligeiramente o enunciado, com a adição de condições de minimalidade, podemos obter o seguinte resultado, cuja prova deixaremos como um exercício. Recordamos que um conjunto S é minimal em relação a uma propriedade, se S satisfaz esta propriedade, mas que nenhum subconjunto próprio de S também satisfaz esta propriedade. Já um conjunto S é mínimo em relação a uma propriedade se não existe nenhum conjunto R com a mesma propriedade que satisfaça $|R| < |S|$.

Teorema 8 Se G é um grafo direcionado e F é um subconjunto de arestas de G então F é um conjunto de retroalimentação minimal se e somente se F é um conjunto minimal cuja reversão torna o grafo acíclico.

Utilizando o Teorema 8, podemos elaborar as regras de redução em termos de reversões de arestas, ao invés de remoção de arestas, uma vez que, se a equivalência entre os dois conjuntos vale para conjuntos minimais então, em particular, ela também vale para conjuntos mínimos.

Um triângulo, em um grafo direcionado G , é um subgrafo H de G com exatamente 3 vértices no qual todo vértice pode ser alcançado a partir de qualquer outro vértice, respeitando-se as orientações das arestas. Apresentamos, agora, as regras de redução.

Regra de redução 4 Se (G, k) é uma instância para o problema do conjunto de arestas de retroalimentação em torneio e f é uma aresta pertencente a pelo menos $k + 1$ triângulos, então podemos reverter f e diminuir k em uma unidade.

Em outras palavras, se uma aresta faz parte de muitos triângulos, então, como devemos destruir todos os ciclos, precisaremos remover ao menos uma aresta de cada triângulo. Como a aresta f faz parte de todos eles, sua remoção destrói todos estes triângulos simultaneamente. Por outro lado, se esta aresta não fosse removida, precisaríamos de pelo menos $k + 1$ remoções de arestas, sendo que só podemos realizar k remoções. Note que o raciocínio aqui é análogo ao da Regra de redução 2 do problema de cobertura de vértices.

Regra de redução 5 Se (G, k) é uma instância para o problema do conjunto de arestas de retroalimentação em torneio e v é um vértice que não faz parte de nenhum triângulo, então podemos removê-lo de G .

Neste caso, como o vértice v não faz parte de nenhum ciclo, v e as arestas incidentes a v são irrelevantes para o problema.

Como veremos com a regra de redução a seguir, as duas regras acima são suficientes para transformar a instância original em uma instância reduzida que forma nosso núcleo, ou ela é maior que um certo tamanho e, neste caso, podemos concluir que se trata de uma instância NÃO.

Regra de redução 6 Se (G, k) é uma instância para o problema do conjunto de arestas de retroalimentação em torneio em que as regras anteriores não podem ser aplicadas, então, se G possui mais de $k(k + 2)$ vértices, então (G, k) é uma instância NÃO.

Para verificar a corretude da regra acima, seja (G, k) uma instância onde as Regras de redução 4 e 5 não podem ser aplicadas. Se esta é uma instância sim, então há no máximo k arestas cuja remoção elimina todos os ciclos de G . Note que cada uma dessas arestas pode fazer parte de no máximo k triângulos, caso contrário uma regra de redução

poderia ser aplicada. Assim, para cada aresta e em uma solução, além das duas extremidades de e , apenas outros k vértices podem formar ciclos com e . Por outro lado, todo vértice deve fazer parte de algum triângulo, caso contrário poderíamos aplicar outra regra de redução. Como os únicos triângulos do grafo são aqueles que passam por alguma das k arestas do conjunto de arestas retroalimentação, e cada uma pode formar triângulos com no máximo $k + 2$ vértices, temos um máximo $k(k + 2)$ vértices em uma instância SIM.

Utilizando estas regras de redução, é possível concluir o seguinte resultado.

Teorema 9 *O problema do conjunto de arestas de retroalimentação em torneios possui um núcleo com no máximo $k^2 + 2k$ vértices.*

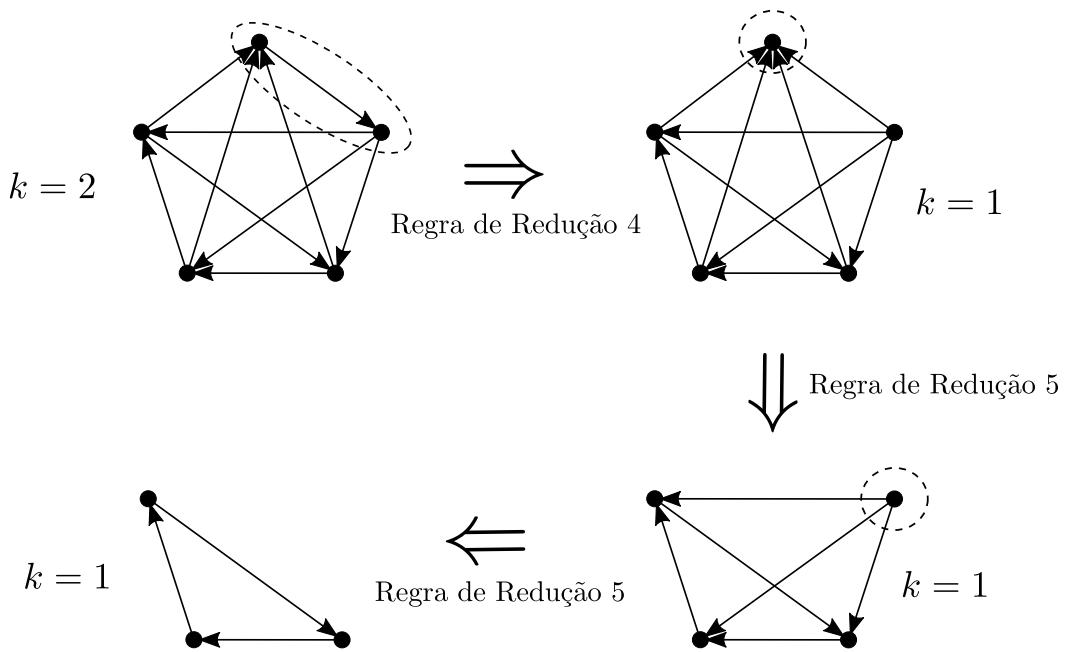


Figure 5.3. Exemplo de aplicação das regras de redução 4 e 5.

5.5.3. Outras técnicas

Assim como no caso das árvores de busca de altura limitada, a obtenção de algoritmos de redução a um núcleo dependem da estrutura do problema em questão. Entretanto, existem algumas técnicas que, embora não possam ser consideradas gerais, por se aplicarem apenas a problemas com determinada estrutura, podem ser bastante úteis na obtenção de núcleos. Como exemplo, podemos mencionar a decomposição em coroa [9], que pode ser utilizada para a obtenção de núcleos não só para problemas em grafos, como o núcleo de tamanho linear para o problema da cobertura de vértices, mas também problemas de outras naturezas, como o problema de satisfabilidade.

Uma evidência da riqueza deste tópico, é o uso de técnicas originadas em outras áreas na obtenção de núcleos, como ferramentas de otimização combinatória, álgebra linear, argumentos probabilísticos, dentre outros.

Finalmente, cabe mencionar uma noção um pouco mais geral de núcleo, denominada *núcleo de Turing*. Em linhas gerais, um núcleo de Turing pode ser visto como um procedimento que, ao invés de achar uma única instância reduzida, como vimos acima, busca a resposta para o problema através da resolução de diversas instâncias reduzidas, todas de tamanho limitado por uma função do parâmetro. Esta noção despertou interesse por permitir a decomposição de um problema em diversos problemas menores, em particular, em casos onde os núcleos tradicionais existentes se mostravam grandes demais.

5.6. Intratabilidade Parametrizada

Além da classe *FPT*, Downey e Fellows definiram classes apropriadas de problemas parametrizados, de acordo com seu nível de intratabilidade parametrizada. Essas classes são organizadas em uma *W-hierarquia* ($FPT \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq W[P]$), e baseadas intuitivamente na complexidade dos circuitos necessários para se verificar a validade de uma solução, ou, alternativamente, na profundidade lógica natural do problema. É conjecturado que cada uma dessas classes são próprias [13, 14, 20], e se $P = NP$ então $FPT = W[P]$ [13].

Para estabelecer que um problema parametrizado é *intratável por parâmetro fixo* necessitamos das seguintes definições adicionais.

Definição 9 [14] *Seja $\Pi(k)$ e $\Pi'(k')$ dois problemas parametrizados, onde $k' \leq g(k)$ para alguma função computável $g : \mathbb{N} \rightarrow \mathbb{N}$. Uma *FPT-redução* (ou transformação paramétrica) de $\Pi(k)$ para $\Pi'(k')$ é uma transformação R tal que:*

1. *Para todo x , temos que $x \in \Pi(k)$ se e somente se $R(x) \in \Pi'(k')$;*
2. *R é computável por um FPT-algoritmo (com relação a k);*

Se uma *FPT-redução* existe entre Π e Π' então Π é transformado (ou se reduz) parametricamente a Π' .

Lema 10 (transitividade) *Dado dois problemas parametrizados Π , Π' and Π'' , se Π se reduz parametricamente a Π' e Π' se reduz parametricamente a Π'' então Π se reduz parametricamente a Π'' .*

Lema 11 (preservação da tratabilidade por parâmetro fixo) *Dado dois problemas Π e Π' , se Π se reduz parametricamente a Π' e Π' é tratável por parâmetro fixo então Π é tratável por parâmetro fixo.*

As seguintes definições fornecem o análogo parametrizado da classe *NP* na teoria de *NP-completude*. Estas classes são, na maior parte, baseadas numa série de circuitos sucessivamente mais poderosos para verificação de uma solução, onde soluções são codificadas como vetores de entrada para estes circuitos e os parâmetros são codificados em pesos destes vetores de entrada [21].

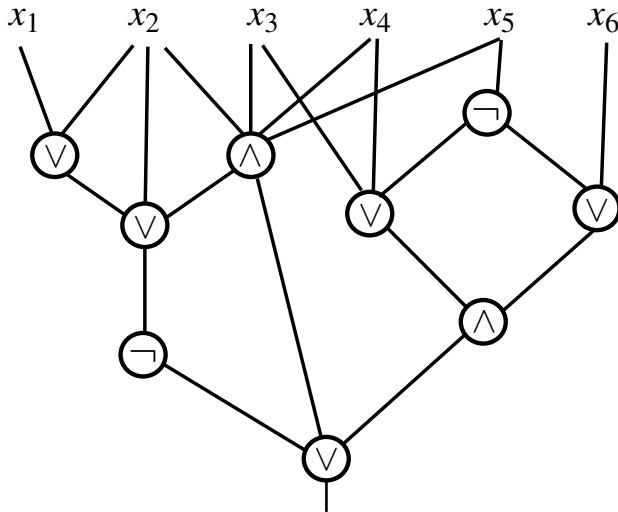


Figure 5.4. Exemplo de circuito booleano de decisão, onde (\wedge) , (\vee) e (\neg) denotam portas lógicas E, Ou e de negação, respectivamente.

Um circuito booleano de decisão consiste de variáveis booleanas de entrada, portas lógicas de negação, portas lógica E, portas lógicas Ou, e uma única porta lógica de saída. Um exemplo de circuito booleano é apresentado na Figura 1.4.

O problema *Satisfabilidade de Circuito* consiste em dado um circuito booleano de decisão C , decidir se existe uma atribuição de valores às variáveis de entrada de C de tal forma que sua saída seja “verdadeiro”.

Definição 10 [13] Seja C um circuito booleano de decisão com variáveis de entrada x_1, \dots, x_n .

- O entrelaçamento de C é definido como o número máximo de portas lógicas largas em qualquer caminho da variável de entrada até a linha de saída (Uma porta é denominada larga se suas entradas excedem algum limite constante, em geral dois).
- A profundidade de C é definida como o comprimento do maior caminho de uma variável de entrada até a linha de saída em C .
- O peso de uma atribuição às variáveis de um circuito booleano C (uma atribuição para C) é o número de 1's nesta atribuição.

Para a próxima definição se faz necessário a seguinte formulação.

Satisfabilidade Ponderada em Circuitos de Entrelaçamento t e Profundidade h WCS(t, h)

Instância: Um circuito de decisão C com entrelaçamento t e profundidade h .

Parâmetro: Um inteiro positivo k .

Questão: C possui uma atribuição satisfatóvel com peso k ?

Definição 11 [13] Um problema parametrizado Π pertence à classe $W[t]$ se e somente se Π se reduz parametricamente a $WCS(t, h)$, para alguma constante h .

De maneira informal, um problema parametrizado Π pertence a classe $W[t]$ se existir uma redução uniforme de Π ao problema de determinar se um dado circuito lógico de decisão C aceita um vetor de entrada de tamanho k , onde C possui em qualquer caminho da entrada até a saída no máximo t portas lógicas de tamanho irrestrito e profundidade no máxima h , para alguma constante h .

Definição 12 (A W-hierarquia) A união destas classes $W[t]$ juntamente com as classes $W[P]$ e XP , denota-se W -hierarquia. $W[P]$ denota a classe obtida por considerar nenhuma restrição sobre profundidade. Portanto, a W -hierarquia é

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq W[P] \subseteq XP.$$

Downey and Fellows conjecturaram que cada uma dessas relações de inclusão na W -hierarquia é própria [13].

Observação 3 Uma alternativa para mostrar que um problema parametrizado Π pertence à classe $W[t]$, $t \geq 1$, é apresentar uma FPT-redução para algum problema parametrizado pertencente a $W[t]$. Veja Lema 10.

Além da W -hierarquia, na complexidade parametrizada há outras hierarquias de classes de problemas parametrizados, tais como a M-hierarquia e A-hierarquia. Entretanto, este curso terá como foco somente a W -hierarquia.

Na complexidade parametrizada, definimos $W[t]$ -dificuldade e $W[t]$ -completude de um problema parametrizado $\Pi(k)$ com relação à classe de complexidade $W[t]$ ($t \geq 1$), como na teoria da complexidade clássica: $\Pi(k)$ é $W[t]$ -difícil sob FPT-reduções se todo problema em $W[t]$ se reduz parametricamente a $\Pi(k)$; $\Pi(k)$ é $W[t]$ -completo sob FPT-reduções se $\Pi(k) \in W[t]$ e $\Pi(k)$ é $W[t]$ -difícil.

5.6.1. Diferentes Níveis de Intratabilidade

Através da teoria da complexidade parametrizada, é possível distinguir diferentes níveis de intratabilidade entre problemas NP-completos, algo que não era possível através da complexidade clássica. Esse refinamento pode ser observado através da classificação das versões parametrizadas naturais destes problemas na W -hierarquia.

Por exemplo. Considere as naturais parametrizações dos seguintes problemas clássicos em grafos.

Conjunto Independente(c)

Instância: Um grafo $G = (V, E)$.

Parâmetro: Um inteiro positivo c .

Questão: G possui um conjunto de vértices I , tal que $|I| \geq c$ e I não contém nenhum par de vértices adjacentes?

Conjunto Dominante(k)

Instância: Um grafo $G = (V, E)$.

Parâmetro: Um inteiro positivo k .

Questão: G possui um conjunto de vértices D , tal que $|D| \leq k$ e todo vértice $v \in V \setminus D$ é adjacente a pelo menos um vértice em D ?

Conjunto Independente. Conforme podemos observar na Figura 1.5, existe uma FPT-redução de Conjunto Independente(c) para WCS(1,3), onde cada vértice v_i de G é representado por uma entrada x_i , tal que toda atribuição que satisfaça o circuito associado C , induz um conjunto independente I para G , e vice-versa, sendo x_i igual a “verdadeiro” se e somente se v_i pertence ao conjunto I . Logo, pela Definição 11, temos que Conjunto Independente(c) \in W[1].

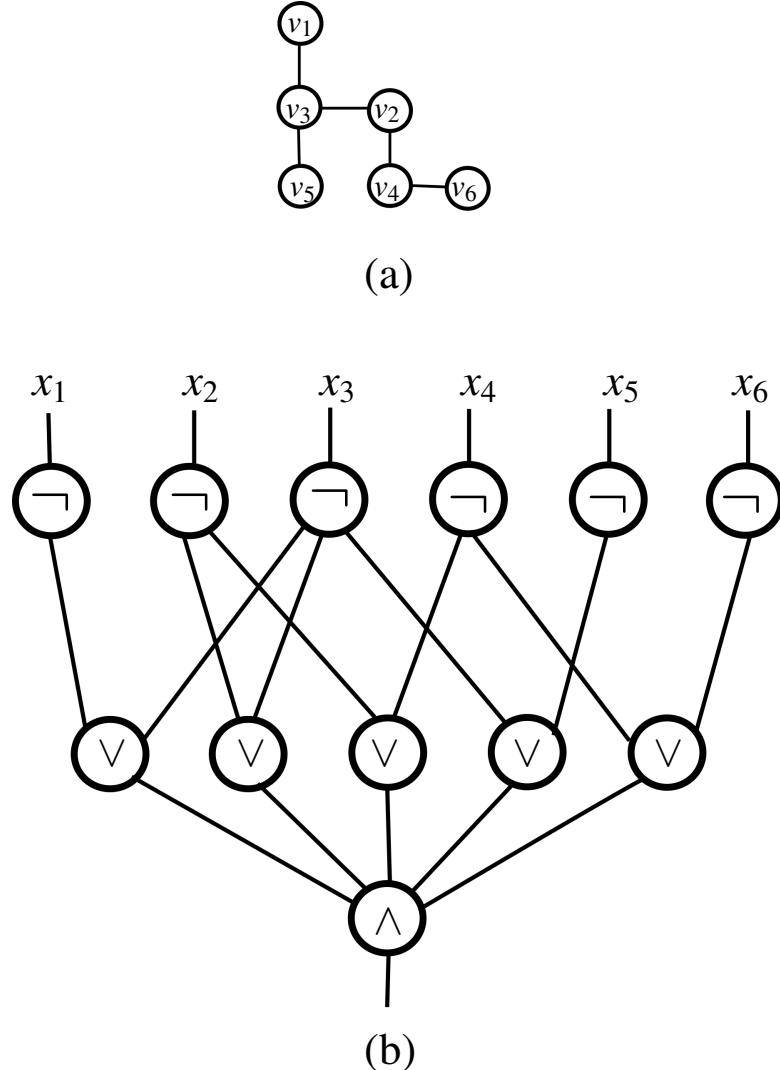


Figure 5.5. (a) Um instância G do problema Conjunto Independente(c); (b) circuito booleano de decisão C associado a G com profundidade três e apenas uma porta lógica larga em qualquer caminho a partir de uma entrada até a saída.

Conjunto Dominante. Para o problema Conjunto Dominante(k) torna-se necessário a construção de um circuito booleano um pouco mais complexo. Conforme Figura 1.6, podemos observar que existe uma FPT-redução de Conjunto Dominante(k) para WCS(2,2), onde cada vértice v_i de G é representado por uma entrada x_i , tal que toda atribuição que satisfaça o circuito associado C , induz um conjunto dominante I para G , e vice-versa, sendo x_i igual a “verdadeiro” se e somente se v_i pertence ao conjunto D . Através de tal

redução temos que Conjunto Dominante(k) $\in \text{W}[2]$.

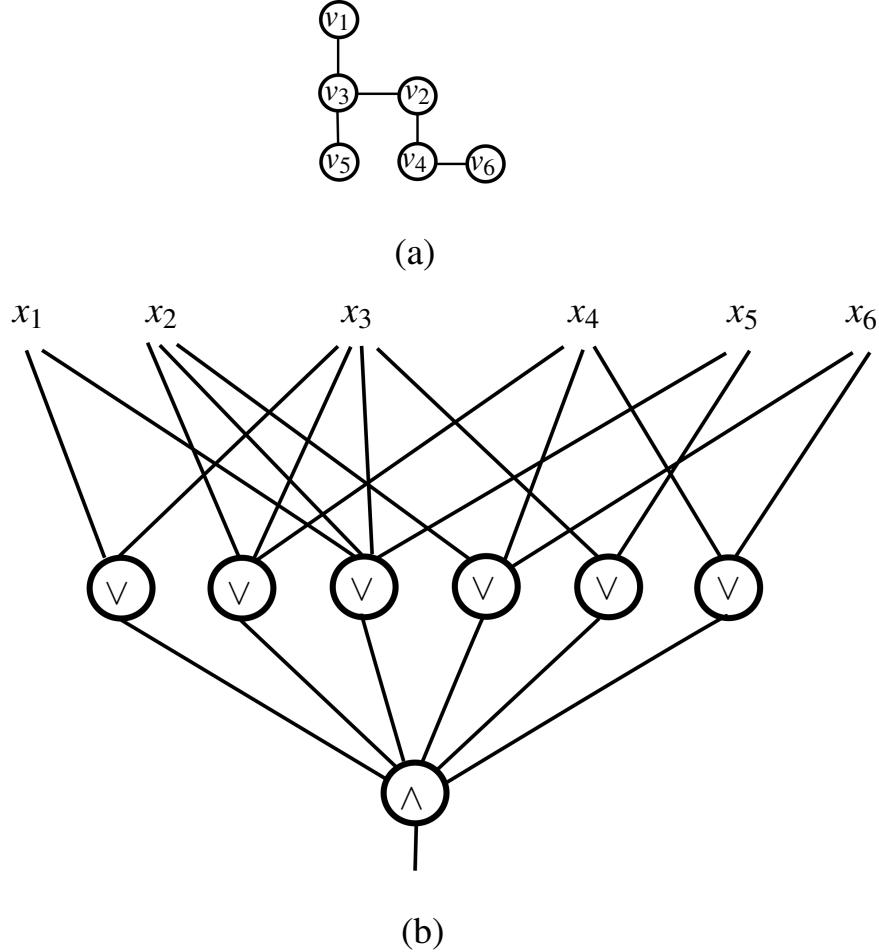


Figure 5.6. (a) Um instância G do problema Conjunto Dominante(k); (b) circuito booleano de decisão C associado a G com profundidade dois e máximo duas portas lógicas largas em qualquer caminho a partir de uma entrada até a saída.

Notem que o circuito booleano de decisão do problema Conjunto Independente(c) utiliza apenas uma porta lógica larga (entrelaçamento um), enquanto o circuito booleano de decisão do problema Conjunto Dominante(k) possui entrelaçamento dois. Poderíamos então questionar, se existe um circuito booleano de decisão para o problema Conjunto Dominante(k) com entrelaçamento menor do que dois, a resposta a esta pergunta é “provavelmente não”, uma vez que Conjunto Dominante(k) é conhecido ser W[2]-completo [13], enquanto Conjunto Independente(c) é W[1]-completo [13]. Deste fato, segue que não existe uma FPT-redução do problema Conjunto Dominante(k) para o problema Conjunto Independente(c), a menos que $\text{W}[1]=\text{W}[2]$ (o que conjectura-se não ser verdade).

O Análogo ao Teorema de Cook

Downey and Fellows [13] demonstraram um teorema análogo ao de Cook exibindo um número de problemas combinatórios que possuem complexidade parametrizada similar a versão parametrizada do problema de Aceitação da Máquina de Turing Não Determinística. Os principais problemas parametrizados são os seguintes.

Aceitação da Máquina de Turing(k)

Instância: Uma máquina de Turing não determinística \mathbb{M} e uma string x .

Parâmetro: Um inteiro positivo k .

Questão: \mathbb{M} aceita x com no máximo k passos?

Este problema é o análogo parametrizado do problema ACEITAÇÃO DA MAQUINA DE TURING (onde o limite no número de passos é polinomial no tamanho da entrada $|x|$, em vez de k), que é o problema NP-completo básico genérico na teoria de complexidade clássica. ACEITAÇÃO DA MÁQUINA DE TURING(k) pode ser trivialmente resolvido em tempo $O(n^{k+1})$, onde n denota o tamanho total da entrada. Isso é feito explorando-se exaustivamente todos os caminhos computacionais de k passos. Acredita-se que este resultado não possa ser significativamente melhorado.

q -CNF Satisfabilidade Ponderada(k)

Instância: Uma expressão booleana F na forma normal conjuntiva (CNF) tal que cada cláusula possui no máximo q literais ($q \geq 2$).

Parâmetro: Um inteiro positivo k .

Questão: F possui um atribuição satisfatível de peso k ?

Teorema 12 [13] (Análogo ao Teorema de Cook) Os seguintes problemas são completos para a classe W[1]:

1. Aceitação da Máquina de Turing(k).

2. q -CNF Satisfabilidade Ponderada(k).

A partir do Teorema 12 diversos outros problemas têm sido mostrados serem W[1]-completos, tais como Clique(k) e Conjunto independente(k), onde k é o parâmetro para o tamanho dos subconjuntos buscados.

O Teorema análogo ao de Cook pode ser utilizado como ferramenta para mostrar a W[1]-dificuldade e W[1]-completude de problemas parametrizados. Pra estabelecer alguns resultados sobre classes W[t] ($t \geq 2$), Downey and Fellows apresentaram um teorema de mais alto nível, o *Teorema da Normalização*.

Definição 13 Uma fórmula booleana F é t -normalizada se F é da forma **produto-de-soma-de-produto...** de literais com t -alternâncias.

Note que a fórmula 2-normalizada é uma fórmula CNF.

Satisfabilidade Ponderada t -Normalizada(k)

Instância: Uma expressão booleana t -normalizada F ($t \geq 2$).

Parâmetro: Um inteiro positivo k .

Questão: F possui um atribuição satisfatível de peso k ?

Teorema 13 [13] (Teorema da Normalização) **Satisfabilidade Ponderada t -Normalizada(k)** é completa para $W[t]$, para todo $t \geq 2$.

Utilizando esse último teorema muitos outros problemas parametrizados foram mostrados serem $W[t]$ -difícil ou $W[t]$ -completo, para algum $t \geq 2$. Por exemplo, o problema parametrizado CONJUNTO DOMINANTE(k) (k é o tamanho do conjunto buscado) foi mostrado ser $W[2]$ -completo. Diversos outros resultados podem ser encontrados em [12, 13, 14, 20].

Como dito anteriormente, temos a relação de inclusão

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \dots ,$$

e existe a conjectura de que cada uma dessas relações é própria. A união das classes da W -hierarquia é denotada por WH . Se $P = NP$, temos então $WH \subseteq FPT$. A classe $W[1]$ é atualmente a classe mais estudada de problemas parametrizados.

5.7. FPT-reduções × Reduções de Tempo Polinomial

Assim como é feito com a complexidade clássica, podemos encontrar evidências para a intratabilidade parametrizada estudando as noções apropriadas de transformações de problemas. A propriedade essencial das transformações paramétricas é que, se Π pode ser transformado em Π' e $\Pi' \in FPT$, então $\Pi \in FPT$. Caso Π seja $W[t]$ -completa, então Π' é $W[t]$ -difícil, sendo completa para a classe $W[t]$ se $\Pi' \in W[t]$.

Nesta seção, ilustraremos alguns exemplos de FPT-reduções, e destacaremos a principal distinção entre FPT-redução e a redução de tempo polinomial (redução de Karp), a qual estamos habituados a utilizar na complexidade clássica.

Nesta seção, consideraremos os seguintes problemas:

1. COBERTURA POR VÉRTICES
2. CONJUNTO INDEPENDENTE
3. CLIQUE
4. CONJUNTO DOMINANTE

Esses quatro problemas clássicos são NP-completos, e portanto são redutíveis dois a dois através de reduções de tempo polinomial. Como exemplo, considere as seguintes reduções de tempo polinomial:

- CLIQUE \propto CONJUNTO INDEPENDENTE

Este é um dos exemplos mais simples de redução de tempo polinomial entre problemas.

Dado uma grafo G construímos o seu grafo complementar \bar{G} , isto é, $V(G) = V(\bar{G})$, $E(G) \cap E(\bar{G}) = \emptyset$ e $E(G) \cup E(\bar{G}) = E(K_n)$, onde K_n é um grafo completo com $V(K_n) = V(G)$.

Neste ponto, basta observar que G possui uma clique de tamanho k se e somente se \bar{G} possui um um conjunto independente de tamanho k . A Figura 1.7 ilustra um grafo G e seu complemento \bar{G} , respectivamente com a clique máxima de G e o conjunto independente máximo de \bar{G} destacados.

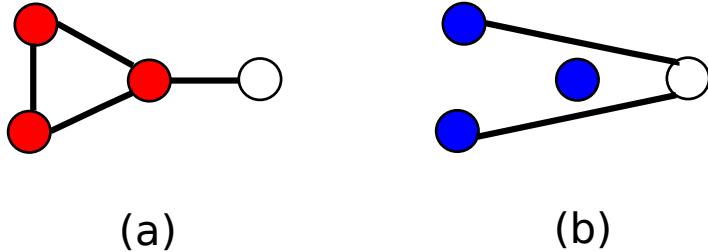


Figure 5.7. (a) Um grafo G e uma clique C de G destacada em vermelho; (b) \bar{G} e o conjunto independente de \bar{G} associado a C em azul.

• CONJUNTO INDEPENDENTE \propto COBERTURA POR VÉRTICES

Um outro exemplo de redução simples é apresentado a seguir.

O problema CONJUNTO INDEPENDENTE facilmente se reduz em tempo polinomial ao problema COBERTURA POR VÉRTICES, pois dado um conjunto independente I de um grafo G , o complemento deste conjunto, $V(G) \setminus I$, é uma cobertura por vértices de G , veja Figura 1.8. Fica como exercício para o aluno verificar a corretude desta afirmação.

Sendo assim, um grafo G possui um conjunto independente de tamanho k se e somente se G possui uma cobertura por vértices de tamanho $n - k$, onde $n = V(G)$.

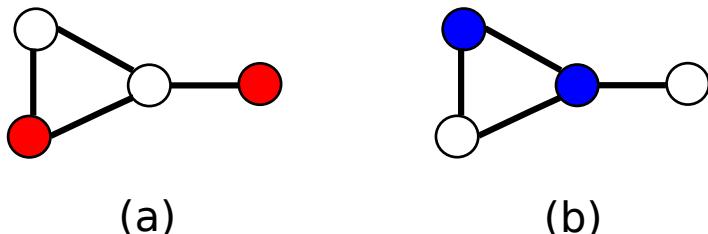


Figure 5.8. (a) Um grafo G e um conjunto independente máximo I de G destacado em vermelho; (b) G e uma cobertura por vértices mínima, $V(G) \setminus I$, destacada em azul.

• COBERTURA POR VÉRTICES \propto CONJUNTO DOMINANTE

Reduzimos COBERTURA POR VÉRTICES ao problema CONJUNTO DOMINANTE da seguinte maneira:

Criamos um grafo G' , onde inicialmente $G' = G$. Para cada aresta $e = (u, v)$ em G' adicionamos um novo vértice z_e , e arestas $(u, z_e), (v, z_e)$ formando triângulos. A Figura 1.9 ilustra um grafo G e o grafo G' associado, onde uma cobertura por vértices mínima de G e um conjunto dominante mínimo de G' encontram-se destacados.

Neste ponto devemos provar que G possui uma cobertura por vértices de tamanho no máximo k se e somente se G' possui um conjunto dominante de tamanho no máximo k .

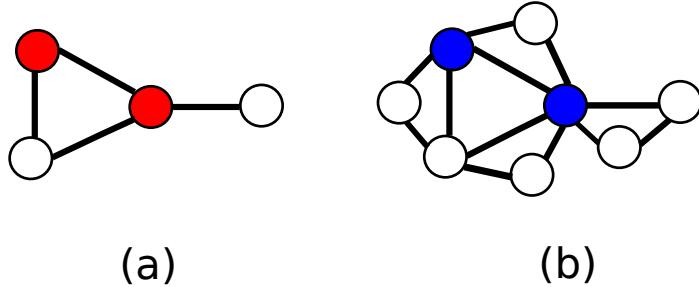


Figure 5.9. (a) Um grafo G e uma cobertura por vértices mínima de G destacada em vermelho; (b) G' e um conjunto dominante mínimo de G' destacada em azul.

Se G possui uma cobertura por vértices S , então o mesmo conjunto é um conjunto dominante de G' , pois por definição de cobertura por vértice, cada aresta em G é incidente a pelo menos um vértice em S , portanto, todo vértice z_e possui pelo menos um vizinho em S .

Por outro lado, se G' possui um conjunto dominante S' então uma cobertura por vértices S de G pode ser obtida, primeiramente fazendo $S = S'$ e em seguida substituindo cada vértice $z_e \in S'$ por um de seus vizinhos em G' . Como todo vértice z_e ou pertence a S' ou possui um vizinho em S' , então toda aresta de G possui pelo menos um vértice em S .

• CONJUNTO DOMINANTE \Leftrightarrow CLIQUE

O problema CONJUNTO DOMINANTE, por sua vez, se reduz em tempo polinomial ao problema CLIQUE da seguinte forma:

Para cada aresta $(u, v) \in G$ criamos dois vértices $[u, v]$ e $[v, u]$ em G' . Para cada dois vértices $[u_1, v_1]$ e $[u_2, v_2]$ de G' adicionamos uma aresta entre eles se $u_1 \neq u_2$, $u_1 \neq v_2$ e $u_2 \neq v_1$. Neste ponto, devemos mostrar que G' possui uma clique de tamanho $n - k$ se e somente se G possui um conjunto dominante de tamanho k , onde $n = |V(G)|$. A Figura 1.10 ilustra um grafo G e o grafo G' associado, respectivamente com o conjunto dominante mínimo de G e a clique máxima de G' destacados.

A intuição é que o vértice $[u, v]$ deve ser interpretado como “ u é dominado por v ”.

Considere um conjunto dominante S em G . Construímos um conjunto S' como segue: para cada vértice $u \notin S$ selecionamos um vizinho arbitrário v de u onde $v \in S$, e adicionamos $[u, v]$ a S' . Claramente S' é uma clique de tamanho $|V(G)| - |S|$.

Por outro lado, considere uma clique S' em G' . Seja S o conjunto de vértices u em $V(G)$ tal que não existe vértice $v \in V(G)$ tal que $[u, v] \in S'$. Observe agora que $|S| = |V(G)| - |S'|$, dado que cada elemento de S' exclui um vértice de G . Considere um vértice $u \notin S$. Deve haver um vértice $v \in V(G)$ tal que $[u, v] \in S'$. Sabemos que $(u, v) \in E(G)$, então basta mostrar que $v \in S$. Suponha que não, então deve haver algum vértice $w \in V(G)$ tal que $[v, w] \in S'$. Mas $[u, v]$ e $[v, w]$ são não adjacentes em G' , contradizendo que S' é uma clique. Portanto S é um conjunto dominante.

As reduções apresentadas acima, demonstram que os quatro problemas listados

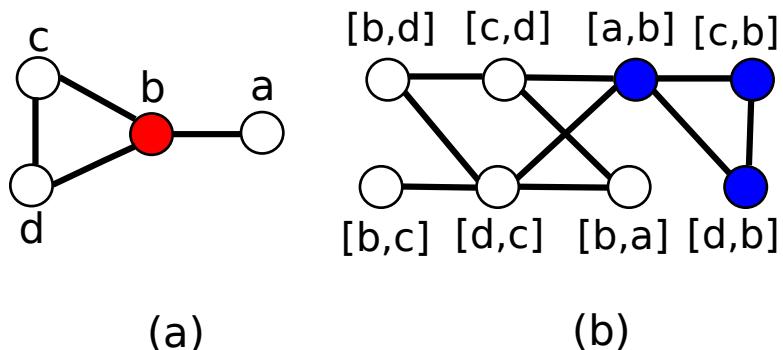


Figure 5.10. (a) Um grafo G e um conjunto dominante mínimo de G destacado em vermelho; (b) G' e uma clique máxima de G' destacada em azul.

se reduzem mutuamente através de reduções de tempo polinomial. Conforme podemos observar na Figura 1.11.

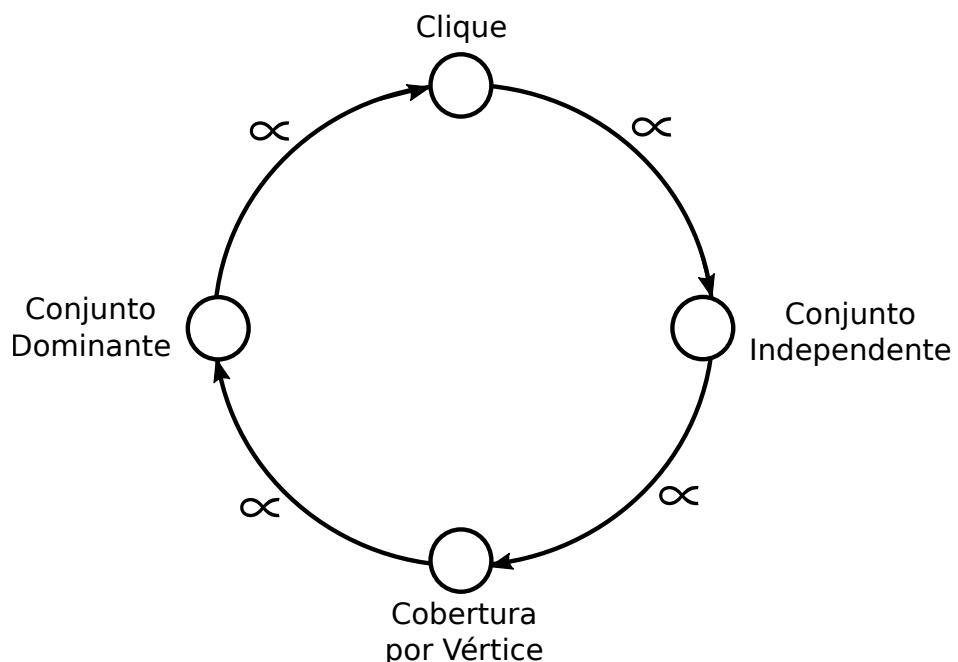


Figure 5.11. Representação das reduções de tempo polinomial apresentadas entre os problemas CLIQUE, CONJUNTO INDEPENDENTE, COBERTURA POR VÉRTICES e CONJUNTO DOMINANTE

Embora as reduções acima possam ser utilizadas para demonstrar a NP-dificuldade dos problemas, algumas destas reduções não podem ser utilizadas para demonstrar a in-tratabilidade parametrizada dos problemas (considerando como parâmetro o tamanho da estrutura buscada).

As principais características que diferem uma FPT-redução de uma redução de tempo polinomial são:

- (a) FPT-reduções podem ser executadas em tempo $f(k).n^{O(1)}$ (onde k é o parâmetro e n

o tamanho da entrada), enquanto reduções de tempo polinomial, como o próprio nome diz, devem ser executadas em tempo polinomial.

Geralmente esta condição não gera dificuldades na construção de uma redução, pois tempo $f(k) \cdot n^{O(1)}$ é uma condição mais relaxada do que tempo polinomial.

- (b) Em FPT-reduções, temos a restrição adicional de que o tamanho dos parâmetros do problema resultante devem ser delimitados puramente por uma função dos parâmetros do problema de origem, isto é, o tamanho dos parâmetros resultantes não podem depender do tamanho do problema original.

Esta condição é a principal diferença entre FPT-reduções e reduções de tempo polinomial. Diversas reduções de tempo polinomial não satisfazem essa condição.

Dentre as quatro reduções de tempo polinomial exibidas nesta seção, temos que:

- A redução de CLIQUE para CONJUNTO INDEPENDENTE é uma FPT-redução.
- A redução de COBERTURA POR VÉRTICES para CONJUNTO DOMINANTE é uma FPT-redução.
- A redução de CONJUNTO INDEPENDENTE para COBERTURA POR VÉRTICES *não* é uma FPT-redução, pois a cobertura por vértices resultante possui tamanho $n - k$, o que não satisfaz a condição (b), uma vez que depende do tamanho da entrada.
- A redução de CONJUNTO DOMINANTE para CLIQUE também *não* é uma FPT-redução, pois a clique resultante possui tamanho $n - k$, o que não satisfaz a condição (b), uma vez que depende do tamanho da entrada.

De fato, FPT-reduções de CLIQUE para CONJUNTO INDEPENDENTE e de COBERTURA POR VÉRTICES para CONJUNTO DOMINANTE eram esperadas, enquanto FPT-reduções de CONJUNTO INDEPENDENTE para COBERTURA POR VÉRTICES e de CONJUNTO DOMINANTE para CLIQUE provavelmente não devem existir pois:

COBERTURA POR VÉRTICES \in FPT

CONJUNTO INDEPENDENTE é W[1]-completo

CLIQUE é W[1]-completo

CONJUNTO DOMINANTE é W[2]-completo

Sendo assim,

- Existe uma FPT-redução de CONJUNTO INDEPENDENTE ou CLIQUE para COBERTURA POR VÉRTICES se e somente se FPT=W[1].
- Existe uma FPT-redução de CONJUNTO DOMINANTE para CONJUNTO INDEPENDENTE ou CLIQUE se e somente se W[1]=W[2].

- Existe uma FPT-redução de CONJUNTO DOMINANTE para COBERTURA POR VÉRTICES se e somente se $FPT=W[1]=W[2]$.

O leitor neste momento poderia estar se perguntando como deveriam ser as FPT-reduções de COBERTURA POR VÉRTICES para CONJUNTO INDEPENDENTE; e de CONJUNTO INDEPENDENTE para CONJUNTO DOMINANTE. Vejamos a seguir.

- **COBERTURA POR VÉRTICES se FPT -reduz a CONJUNTO INDEPENDENTE**

Uma FPT-redução de COBERTURA POR VÉRTICES para CONJUNTO INDEPENDENTE pode ser facilmente obtida, pois COBERTURA POR VÉRTICES $\in FPT$. Assim, podemos para cada instância G de COBERTURA POR VÉRTICES solucioná-la em tempo FPT, criando em seguida uma instância para CONJUNTO INDEPENDENTE que possui solução de tamanho k se e somente se a instância G de COBERTURA POR VÉRTICES possui solução de tamanho k .

Para ilustrar uma FPT-redução de CONJUNTO INDEPENDENTE para CONJUNTO DOMINANTE, iremos utilizar um passo intermediário. Considere o seguinte problema.

Conjunto Independente Multi-Colorido

Instância: Um grafo $G = (V, E)$ com uma k -coloração de vértices.

Parâmetro: o inteiro positivo k .

Questão: G possui um conjunto independente de vértices I contendo um vértice de cada classe de cor?

- **CONJUNTO INDEPENDENTE se FPT -reduz a CONJUNTO INDEPENDENTE MULTI-COLORIDO.**

Dado uma instância G COBERTURA POR VÉRTICES, construímos um grafo G' criando k cópias v_1, v_2, \dots, v_k para cada vértice v e colorindo v_i com cor i . Adicionamos então uma aresta em G' entre dois vértices u_i e v_j , $i \neq j$ se e somente se $u = v$ ou u e v são adjacentes em G . Neste ponto, é fácil ver que G' possui um conjunto independente de tamanho k se e somente se G possui um conjunto independente multi-colorido de tamanho k .

- **CONJUNTO INDEPENDENTE MULTI-COLORIDO se FPT -reduz a CONJUNTO DOMINANTE.**

Seja G um grafo com seus vértices sendo k -coloridos. Construímos um grafo H tal que G possui um conjunto independente multi-colorido se e somente se H possui um conjunto dominante de tamanho k .

Seja (V_1, \dots, V_k) os conjuntos de vértices de G , onde V_i é o conjunto de vértices coloridos com a cor i . Construímos um grafo H como segue:

- para todo vértice $v \in V(G)$, inserimos v em H ;
- para todo $1 \leq i \leq k$, fazemos o conjunto V_i em H ser uma clique através de adição de arestas;
- para todo $1 \leq i \leq k$, adicionamos dois novos vértices x_i, y_i em H e fazemos eles adjacentes a todo vértice em V_i ;

- para toda aresta $e \in E(G)$ com extremidades $u \in V_i$ e $v \in V_j$, criamos um vértice w_e em H e o fazemos adjacente a todo vértice em $(V_i \cup V_j) \setminus \{u, v\}$.

Mostraremos que G possui um conjunto independente k -colorido se e somente se H possui um conjunto dominante de tamanho k .

Seja $Z \subset V(G)$ um conjunto independente tal que $Z = \{z_1, z_2, \dots, z_k\}$ e $z_i \in V_i$. Agora mostraremos que Z é um conjunto dominante em H . x_i e y_i são dominados por z_i . Todos os vértices em V_i são dominados por z_i . Agora, suponha $w_{\{u, v\}}$ com $u \in V_i$ e $v \in V_j$ não possuir um vizinho em Z , isto implica que $u = z_i$ e $v = z_j$, o que implica que z_i e z_j possuem uma aresta em G , o que é uma contradição.

Agora, assuma que temos um conjunto dominante D de tamanho k em H . Como todos os x'_i 's e y'_i 's devem ser dominados, sem perda de generalidade cada V_i deve ter pelo menos um de seus membros em D . Portanto, podemos escrever D como $D = \{d_1, d_2, \dots, d_k\}$ tal que $d_i \in V_i$.

Nos resta mostrar que D é um conjunto independente de tamanho k de G . Claramente os vértices em D são unicamente coloridos pois pertencem a diferentes V'_i 's. Suponha que d_i e d_j possuem uma aresta entre eles em G . Isto implica que w_{d_i, d_j} não possui um vizinho em D , o que é uma contradição.

5.8. Pré-processamento e Inviabilidade de Núcleos Polinomiais

Pré-processamento (ou redução de dados) para reduzir o tamanho da instância é uma das heurísticas mais comumente implementadas na prática para resolver os problemas computacionalmente difíceis. No entanto, um estudo teórico sistemático deles permaneceu uma incógnita até recentemente. Uma das razões para isto é que, se uma entrada para um problema NP-difícil pode ser processada em tempo polinomial para um equivalente de um tamanho muito menor (constante), em geral, em seguida, o algoritmo de pré-processamento pode ser combinado para, na verdade, resolver o problema em tempo polinomial provando que $P = NP$, o que é considerado improvável. No entanto, a situação em matéria de estudo sistemático mudou drasticamente com o advento da complexidade parametrizada. A complexidade parametrizada fornece um quadro natural para analisar algoritmos de pré-processamento. Em um problema com parâmetros, cada instância x vem com um inteiro positivo, ou parâmetro, k . O problema é dito admitir um núcleo (kernel) se, em tempo polinomial, puder reduzir o tamanho da instância x a uma função de k , preservando ao mesmo tempo a resposta.

A noção central na complexidade parametrizada é a tratabilidade por parâmetro fixo (FPT), que é a noção de solucionabilidade em tempo $f(k).p(|x|)$ para qualquer instância (x, k) , onde f é uma função arbitrária do parâmetro k e p é uma função polinomial no tamanho de entrada $|x|$. Resultados teóricos mostram que um problema parametrizado Π é tratável por parâmetro fixo se e somente se existe uma função computável $g(k)$ de tal forma que Π admite um núcleo de tamanho $g(k)$, i.e., algoritmos de pré-processamento que reduzem a entrada em uma instância de tamanho $g(k)$. No entanto, os núcleos obtidos por estes resultados teóricos são geralmente de tamanhos exponenciais (ou até pior) em relação ao parâmetro, enquanto reduções de dados específicos do problema muitas vezes alcançam núcleos quadráticos ou mesmo de tamanho linear. Assim, uma pergunta

natural para qualquer problema concreto em FPT é se tal problema admite redução a um núcleo (kernelization) de tempo polinomial para um núcleo do problema que, no pior caso é delimitada por uma função polinomial do parâmetro. Apesar de várias tentativas, há alguns problemas tratáveis por parâmetro fixo que somente se conhecem núcleos de tamanho exponenciais. Uma explicação foi fornecida em artigo por Bodlaender et al. (2009) [4], onde foi mostrado que a menos que $cNP \subseteq NP/poly$, há problemas tratáveis por parâmetro fixo que não pode ter um núcleo de tamanho polinomial. Isso desencadeou novos trabalhos mostrando limites para algoritmos de pré-processamento (algoritmos de redução a um núcleo ou algoritmos de kernelização).

5.8.1. Núcleos Polinomiais

Algoritmos de redução a um núcleo (Kernelização) são algoritmos de pré-processamento que satisfazem algumas condições adicionais. Observe que kernelização é uma redução de tempo polinomial de um problema para ele mesmo com a propriedade adicional que o tamanho da imagem é delimitado em termos do parâmetro do argumento. De fato, tratabilidade parametrizada e kernelização possuem noções idênticas. O teorema central para o entendimento entre kernelização e tratabilidade parametrizada pode ser encontrado em [13, 14, 20].

Teorema 14 *Para todo problema parametrizado Π , as seguintes afirmações são equivalentes:*

1. $\Pi \in FPT$.
2. Π é decidível e Π admite um algoritmo de kernelização.

Tendo que FPT é a classe de problemas redutíveis a um núcleo, temos agora uma outra maneira de analisar mais profundamente esta classe. Por exemplo, utilizando o tamanho dos núcleos obtidos como base para classificação.

Um procedimento de redução a um núcleo é usualmente descrito como uma coleção de regras de redução, que são desenvolvidas para transformar as instâncias e preservar algumas propriedades. Cada regra é geralmente aplicada recursivamente, onde cada aplicação encolhe a instância de alguma maneira, onde espera-se ser possível provar que se as regras forem aplicadas até não ser possível aplicá-las então a instância resultante deve ser um núcleo.

Naturalmente, gostaríamos que o núcleo produzido fosse o menor possível a ser obtido por um algoritmo de kernelização. Isto motiva a noção de núcleos polinomiais.

Núcleo Polinomial. Um núcleo polinomial é um núcleo cujo tamanho é polinomial em relação ao parâmetro original.

Conforme vimos anteriormente, o problema COBERTURA POR VÉRTICES parametrizado pelo tamanho da cobertura buscada é um exemplo de problema que admite núcleo polinomial.

5.8.2. Nenhum núcleo polinomial

Agora, estamos prontos para examinar as circunstâncias sob as quais não esperamos um núcleo polinomial para o problema parametrizado.

Esta seção descreve a caracterização de problemas que provavelmente não admitem núcleo de tamanho polinomial, utilizando como ferramenta uma certa propriedade. Esta propriedade tornou-se uma ferramenta muito útil, pois para exibir que um problema provavelmente não admite núcleo polinomial, basta demonstrar o problema em questão satisfaz esta propriedade. O resultados destas seção baseiam-se nos trabalhos seminais de Bodlaender et al. [4] e Fortnow e Santhanam [15].

Enquanto resultados positivos sobre a existência de núcleos polinomiais têm aparecido regularmente nas duas últimas décadas, os primeiros resultados estabelecendo a inviabilidade de núcleo polinomiais para problemas específicos têm aparecido somente recentemente. Em particular, Bodlaender et al. [4] e Fortnow e Santhanam [15] desenvolveram um framework baseado na noção de *composicionalidade*, para mostrar que um problema não admite um núcleo polinomial a menos que $NP \subseteq coNP/poly$, implicando um colapso na hierarquia polinomial até o terceiro nível ($PH = \Sigma_p^3$), que é considerado improvável.

Um *algoritmo de Ou-destilação* para uma dado problema é concebido para atuar como um “operador booleano Ou de instâncias de um problema”, ele recebe como entrada uma sequência de instâncias do problema, e produz uma instância-SIM se e somente se pelo menos uma das instâncias da sequência dada também é uma instância-SIM. Embora seja permitido executar o algoritmo em tempo polinomial em relação ao tamanho total da sequência de entrada, a saída do algoritmo deve ser uma instância cujo tamanho é polinomialmente limitado pelo tamanho da maior instância da sequência de entrada. Formalmente, temos o seguinte:

Definição 14 (*Ou-destilação* [4]) *Seja Π um problema NP-completo. Uma Ou-destilação de Π é um algoritmo de tempo polinomial D que recebe como entrada uma série de m instâncias de Π , e retorna uma outra instância de Π , tal que*

- *Se D possui como entrada m instâncias, cada uma de tamanho no máximo n , então D utiliza tempo polinomial em m e n , e sua saída é delimitada por uma função que é polinomial em n .*
- *Se D têm como entrada instâncias x_1, \dots, x_m , então $D(x_1, \dots, x_m) \in \Pi$ se e somente se $\exists_{1 \leq i \leq m} x_i \in \Pi$.*

Teorema 15 ([4]) *Se um problema NP-completo possui um algoritmo de Ou-destilação então $NP \subseteq coNP/poly$.*

Combinado com o teorema de Yap [22] ($NP \subseteq coNP/poly \Rightarrow PH \subseteq \Sigma_3^P$), temos que uma Ou-destilação para um problema NP-completo também implica em um colapso da hierarquia polinomial até o terceiro nível.

Definição 15 (*Ou-composição* [4]) *Seja $\Pi \subseteq \Sigma^* \times N$ um problema parametrizado. Uma Ou-composição de Π é um algoritmo de tempo polinomial D que recebe como entrada*

uma sequencia $((x_1, k), (x_2, k), \dots, (x_m, k))$, com cada $(x_i, k) \in \Sigma^* \times N$, e retorna um par (x', k') , tal que

- o algoritmo utiliza tempo polinomial em $\sum_{1 \leq i \leq m} |x_i| + k$;
- k' é delimitado por um polinomio em k ;
- $(x', k') \in \Pi$ se e somente se $\exists_{1 \leq i \leq m} (x_i, k) \in \Pi$.

Um problema parametrizado é ou-composicional se ele admite um algoritmo de ou-composição. Se a versão parametrizada de um problema NP-completo admite tanto uma ou-composição quanto um núcleo polinomial, então o problema NP-completo possui uma ou-distilação. Isto implica que se um problema parametrizado possui um algoritmo de ou-composição, então ele não pode admitir núcleo polinomial, a menos que $NP \subseteq coNP/poly$, devido ao Theorem 15.

Teorema 16 ([4]) *Seja $\Pi(k)$ um problema parametrizado ou-composicional tal que Π é NP-completo. Se Π possui núcleo polinomial, então Π também possui um algoritmo de ou-distilação.*

Algoritmos que compõem multiplas instâncias de um problema em uma única instância têm sido desenvolvidas para vários problemas [4, 19, 5, 6]. Exemplos de técnicas de ou-composição podem ser encontradas em [19]: composição por união disjunta, composição utilizando IDs, composição com cores e IDs, e composição como programação dinâmica.

Exemplo de Ou-composição.

Considere o seguinte problema:

Caminho(k)

Instância: Um grafo $G = (V, E)$.

Parâmetro: Um inteiro não negativo k .

Questão: G possui um caminho de comprimento k ?

Este problema é conhecido pertencer a FPT com tempo de execução $2^{O(k)}n^{O(1)}$ via uma técnica conhecida como “color coding” [1].

A seguir apresentamos uma ou-composição para o problema CAMINHO(k). Considere as seguintes instâncias de entrada:

$$(G_1, k), \dots, (G_t, k).$$

Uma algoritmo de ou-composição é trivial, ele simplesmente fornece a união disjunta G' de todos os grafos de entrada como saída. Note que esta operação consome tempo linear, e não altera o valor do parâmetro. Claramente, G' possui um caminho de comprimento no máximo k se e somente se existe i , $1 \leq i \leq t$, tal que G_i possui um caminho de comprimento no máximo k . Logo CAMINHO(k) não admite núcleo polinomial a menos que $NP \subseteq coNP/poly$.

A Ou-composição foi a primeira técnica a ser desenvolvida para demonstrar que alguns problemas em FPT não admitem núcleo polinomial, a menos que $NP \subseteq coNP/poly$. Após o seu surgimento, outras técnicas também foram desenvolvidas tais como PPT-reduções e composição-cruzada (cross-composition). [12, 9]

5.9. Complexidade Parametrizada × Algoritmos de Aproximação

Conforme [2], um problema NP de otimização (em NPO) é formalmente definido como uma 4-tupla $(I, sol, custo, meta)$, onde

- I é um conjunto de instâncias;
- para uma instância $x \in I$, $sol(x)$ é o conjunto de soluções viáveis de x . O tamanho de cada $y \in sol(x)$ é polinomialmente delimitado em $|x|$, e dado x e y , pode ser decidido em tempo polinomial se $y \in sol(x)$;
- dado uma instância x e uma solução viável y , $custo(x, y)$ é uma função inteira positiva computável em tempo polinomial;
- meta é ou *min* ou *max*;

O objetivo é encontrar uma solução viável y que alcança o melhor valor objetivo, isto é,

$$cost(x, y) = goal\{cost(x, y') : y' \in sol(x)\}.$$

O custo da solução ótima para uma instância x é denotada por $opt(x)$. Se y é uma solução para a instância x , então a razão de desempenho de y , $R(x, y)$, é definida como

$$\begin{aligned} cost(x, y)/opt(x) \text{ se } goal &= min, \\ opt(x)/cost(x, y) \text{ se } goal &= max. \end{aligned}$$

Assim, $R(x, y)$ é sempre pelo menos 1; quanto mais próximo de 1, mais próximo a solução está do ótimo. Para um número real $c > 1$, dizemos que uma algoritmo é c -aproximado, se sempre produz uma solução com razão de desempenho no máximo c .

Dizemos que um problema X admite um esquema de aproximação de tempo polinomial (PTAS) se para todo $\varepsilon > 0$, existe um $(1 + \varepsilon)$ -algoritmo polinomial para X . Mais precisamente, existe um algoritmo que produz uma solução arbitrariamente boa. Isto é, um algoritmo A tal que dado uma instância x de X e um $\varepsilon > 0$, produz uma solução $(1 + \varepsilon)$ -aproximada em tempo $|x|^{f(1/\varepsilon)}$ para alguma função f . Claramente, tal algoritmo é executado em tempo polinomial para todo valor fixo de ε .

Se ε é pequeno então o expoente do polinômio $|x|^{f(1/\varepsilon)}$ pode ser muito grande. Um esquema de aproximação de tempo polinomial eficiente (EPTAS) é um PTAS com tempo de execução da forma $f(1/\varepsilon).|x|^{O(1)}$.

Como sabemos, muitas vezes é mais fácil trabalhar com problemas de decisão ao invés de problemas de otimização, além disso a teoria da complexidade é baseada em problemas de decisão. No entanto, há uma forma padrão na qual problemas de otimização podem ser transformados em problemas de decisão equivalente. Se X é um problema de minimização (ou maximização), então definimos um problema de decisão como: uma instância x de X , um inteiro k e a seguinte questão “ $opt(x) \leq k?$ ” (ou “ $opt(x) \geq k?$ ”).

Se X é um problema de otimização, a parametrização padrão de X é o problema parametrizado por k . Se a parametrização padrão é tratável por parâmetro fixo, então isto significa que temos um algoritmo eficiente para determinar o ótimo para instâncias onde o ótimo é pequeno.

Teorema 17 (*Bazgan [3]; Cesati e Trevisan [7]*) *Se um problema de otimização em NPO, X , admite EPTAS, então sua parametrização padrão é tratável por parâmetro fixo.*

Prova. Assuma que temos um algoritmo que produz uma solução $(1 + \varepsilon)$ -aproximada em tempo $f(1/\varepsilon).|x|^{O(1)}$. Dado uma instância (x, k) da versão parametrizada padrão de X , assumimos $\varepsilon = 1/(k+1)$, e utilizamos o EPTAS para encontrar uma solução $(1 + \varepsilon)$ -aproximada em tempo $f(1/\varepsilon).|x|^{O(1)} = f(k+1).|x|^{O(1)}$. Sem perda de generalidade, assuma que X é um problema de minimização. Se o ótimo é no máximo k , então o custo de uma solução aproximada é no máximo $(1 + \varepsilon)k = k + (k/k+1) < k + 1$. Como o custo é um inteiro, o custo da solução aproximada é no máximo k . Se o ótimo é maior do que k , então o custo da solução aproximada é também maior que k . Portanto, checando se o custo da solução aproximada é no máximo k , podemos decidir se o ótimo é no máximo k .

Podemos utilizar a contrapositiva para mostrar que é improvável que um problema particular admite um EPTAS.

Corolário 18 *Se a parametrização padrão de um problema de otimização é W[1]-difícil, então o problema de otimização não admite um EPTAS (a menos que FPT = W[1]).*

Observe que o inverso do Teorema 17 não é verdade. Por exemplo, determinar a cobertura mínima por vértices pertence a FPT, mas por outro lado é APX-difícil, e portanto não admite um PTAS. Sendo assim, Teorema 17 possui aplicabilidade limitada.

5.10. Considerações finais e leitura adicional

Esperamos, com este texto, poder ter fornecido uma visão inicial a área de complexidade parametrizada e, com isso, despertar maior interesse por este tópico. Se por um lado esta é uma área de pesquisa ainda em crescimento, por outro lado já pode ser considerada uma área consolidada, com resultados importantes e algumas técnicas bem compreendidas pela comunidade.

Embora os primeiros resultados na área de complexidade parametrizada datem de cerca de 30 anos atrás, a consolidação da área se deu após a publicação do primeiro livro dedicado exclusivamente a este tópico, escrito por Downey e Fellows. Embora já desatualizado, uma vez que retrata o estado da área até 1999, o texto ainda serve de texto

introdutório para um primeiro contato com área [13]. Mais recentemente, em 2013, uma nova versão do livro foi lançada [12], e fornece uma visão atualizada da área, embora diversos tópicos não sejam cobertos em profundidade, devido à abrangência do livro.

Dois outros livros, publicados em 2006, de Flum e Grohe [14] e de Niedermeier [20], fornecem possibilidades interessantes de aprofundamento na área. Em particular, o livro de Flum e Grohe é mais focado em aspectos de complexidade e lógica, e pode ser mais adequados para os leitores com interesse nestas áreas.

Uma outra referência interessante é o livro de Cygan et al [9], em fase de elaboração. Ao contrário dos livros de Downey e Fellows, este livro abre mão de uma abrangência exageradamente grande, em troca de uma apresentação mais didática.

Acreditamos que estas fontes sejam referências adequadas para o próximo passo no aprofundamento do conhecimento daqueles que se interessarem pelo tema.

References

- [1] Noga Alon, Raphael Yuster, Uri Zwick, Color-coding, *J.ACM*, v. 42(4), 844-856, 1995.
- [2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, *Complexity and Approximation*, Springer-Verlag, Berlin, 1999.
- [3] C. Bazgan. Schémas d'approximation et complexité paramétrée. Technical report, Université Paris Sud, 1995.
- [4] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, Danny Hermelin, On problems without polynomial kernels, *Journal of Computer and System Sciences*, v. 75, p. 423-434, 2009.
- [5] Hans L. Bodlaender, Stéphan Thomassé, Anders Yeo, Kernel bounds for disjoint cycles and disjoint paths, *Theoretical Computer Science*, v. 412, p. 4570-4578, 2011.
- [6] Hans L. Bodlaender, Bart M.P. Jansen, Stefan Kratsch, Kernel bounds for path and cycle problems, *Theoretical Computer Science*, doi:10.1016/j.tcs.2012.09.006, 2012.
- [7] M. Cesati, L. Trevisan. On the efficiency of polynomial time approximation schemes. *Inform. Process. Lett.*, v. 64(4), p. 165-171, 1997.
- [8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. MIT Press, Terceira edição, 2009.
- [9] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk and Saket Saurabh. *Parameterized Algorithms*. A ser publicado, Springer, 2015+.
- [10] Rodney G. Downey, Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *Siam Journal on Computing*, v. 24, n. 4, p. 873-921, ago. 1995.
- [11] Rodney G. Downey, Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, v. 141, p. 109-131, abr. 1995.
- [12] Rodney G. Downey, Michael R. Fellows. *Fundamentals of Parameterized Complexity*, Springer, 2013.
- [13] Rodney G. Downey, Michael R. Fellows. *Parameterized complexity*, Monographs in Computer Science, Springer, 1999.
- [14] Jörg Flum; Martin Grohe. *Parameterized complexity theory*, Springer, 2006.
- [15] L. Fortnow, R. Santhanam, Infeasibility of instance compression and succinct PCPs for NP, *Journal of Computer and System Sciences*, v.77, p. 91-106, 2011.
- [16] Michael R. Garey, David S. Johnson. *Computer and intractability. A Guide to the NP-Completeness*. Ney York, NY: WH Freeman and Company, 1979.

- [17] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 1972.
- [18] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research* **8**(4), 538?548 (1983).
- [19] Neeldhara Misra, Venkatesh Raman, Saket Saurabh, Lower Bounds on Kernelization, *Discrete Optimization*, v. 8, p. 110-128, 2011.
- [20] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, 2006.
- [21] H. T. Wareham. Systematic parameterized complexity analysis in computational phonology, PhD thesis, University of Victoria, 1999.
- [22] Chee-Keng Yap, Some Consequences of Non-Uniform Conditions on Uniform Classes, *Theoretical Computer Science*, v. 26, p. 287-300, 1983.

Capítulo

6

Simulação de Robôs Móveis e Articulados: Aplicações e Prática

Rafael Alceste Berri, Valdir Grassi Jr. e Fernando Santos Osório

Abstract

This course aims to provide an introduction to key concepts of mobile robotics and articulated robots, through simulations and based on practical examples presented in the course. Practices will be made available to the participants. The R&D of Intelligent Mobile Robots is an important academic and industrial field nowadays, which is directly related to mobile robots design, autonomous vehicles, humanoid robots, industrial manipulator arms and aerial robots. The tool we adopted in this short-course allows simulating different types of mobile and articulated robots, as well as, simulates sensors and other elements of the virtual environment. In this short-course different concepts about mobile and intelligent robots will be addressed: sensors, actuators, robot configuration (kinematic models), and intelligent robot behavior programming, that is, the implementation of behaviors integrating the robot perception-decision-action cycle. This simulator tool, V-REP PRO (Free and Non-Limited Educational Version), lets the user play with realistic situations (high degree of 3D realism including physical simulation), and also allows testing different models of well-known robots and sensors. This short-course will provide a direct contact with robotics and will open several different opportunities for new studies and research development in this field, once the simulator reproduces quite closely real-world robotic applications.

Resumo

Este curso visa apresentar uma introdução aos principais conceitos da área de robótica móvel e sobre robôs articulados, através de uma abordagem baseada em simulações e orientada a exemplos práticos apresentados no curso, os quais serão disponibilizados e poderão ser testados pelos participantes. A robótica móvel inteligente é uma importante área em grande expansão na atualidade, onde está diretamente relacionada ao projeto de robôs móveis, veículos autônomos, robôs humanoides, braços manipuladores e robôs

aéreos. A ferramenta adotada neste mini-curso permite simular diferentes tipos de robôs móveis e articulados, além de sensores e outros elementos que podem compor o ambiente virtual de simulação montado pelo usuário. Neste mini-curso serão abordados os conceitos sobre sensores, atuadores, configuração de robôs (modelos cinemáticos), e sobre a programação de comportamentos inteligentes, ou seja, a implementação de comportamentos integrando a percepção-decisão-ação do robô. O simulador adotado permite reproduzir situações reais com um alto grau de realismo 3D, além de permitir a realização de testes com variados modelos de robôs e de sensores. O curso provê um contato direto com a robótica e com inúmeras possibilidades de desenvolvimentos de estudos e pesquisas, usando um simulador que reproduz de modo bastante fiel as aplicações robóticas do mundo real.

6.1. Introdução

A palavra robô é originária da palavra tcheca *robota* que significa literalmente “trabalhador humilde” [Romero et al. 2014a]. Sua primeira aparição foi em uma peça teatral de Karel Capek intitulada R.U.R. (*Rossum’s Universal Robots*), de 1921, onde Rossum, um inventor, cria trabalhadores artificiais (os *roboti*), aptos a desempenhar algumas tarefas humanas do quotidiano.

Da ficção até os dias atuais, a robótica tem avançado significativamente. Inúmeros novos recursos de *hardware* e *software* vem aparecendo e facilitando a sua ampla disseminação em nossa sociedade, tornando-se cada vez mais difícil desvincular os robôs do futuro da humanidade. Em 2007, Bill Gates em seu artigo publicado na *Scientific American* [Gates 2007], comparou o crescimento da robótica com a disseminação do computador nos lares, onde faz a alusão de que “PCs irão deixar o seu lugar em cima das mesas para passar a ver, ouvir, tocar e manipular objetos”.

Os robôs móveis (com capacidade de locomoção pelo ambiente) e robôs articulados (possuem uma base fixa), sejam eles projetados para executar tarefas repetitivas (“não inteligentes”), para serem teleoperados, ou para operar de forma autônoma e inteligente, constituem uma importante área de pesquisa e desenvolvimento em grande expansão na atualidade [Department 2014]. O desenvolvimento de pesquisas e projetos de robôs móveis tem resultado em produtos e empresas com grande crescimento. Este avanço robótico já se apresenta a disponibilidade de robôs nas mais diversas formas e tipos. Na Figura 6.1 são mostrados exemplos de alguns robôs, onde, o ACM-R5 [HiBot 2015] da *HiBot* (Figura 6.1(a)) é um robô serpente que consegue operar em baixo da água, sendo já utilizado em filmes de *Hollywood*; o humanóide NAO [Aldebaran Robotics 2015] (Figura 6.1(b)) desenvolvido pela Aldebaran Robotics e o Pioneer p3dx [Adept MobileRobots 2015] (Figura 6.1(c)) são grandemente utilizados em pesquisas científicas; o braço robótico IRB 140 [ABB 2015a] (Figura 6.1(d)) é para uso industrial da ABB; tratando-se de ambientes domésticos, o Roomba [iRobot 2015] (Figura 6.1(e)) da iRobot é um aspirador de pó robótico e o Landroid Wg794e [Landroid 2015] (Figura 6.1(f)) é um cortador de grama autônomo; e, mencionando robô de muitas utilidades no espaço, pode-se citar o Robonaut 2 (Figura 6.1(g)) e os rovers de exploração de Marte da Nasa [NASA 2015].

Dentro deste contexto, é de grande impacto econômico e de grande relevância social e industrial, o desenvolvimento da robótica no Brasil, para permitir que mais pessoas



Figura 6.1. Exemplos de Robôs: o (a) ACM-R5 [HiBot 2015], o (b) NAO [Aldebaran Robotics 2015] e o (c) Pioneer p3dx [Adept MobileRobots 2015], (d) IRB 140 [ABB 2015a], o (e) Roomba [iRobot 2015], (f) Landroid Wg794e [Landroid 2015] e o (g) Robonaut 2 [NASA 2015].

possam fazer parte dessa revolução. As pesquisas e desenvolvimentos em robótica envolvem diversas áreas, tais como: Computação (*software*), Engenharia Eletrônica (*hardware*) e Mecânica (projeto da estrutura do robô), entre outras áreas de conhecimento (p.ex. Física, Matemática). A computação tem um papel muito importante, pois provê o suporte necessário para possibilitar a criação de sistemas de controle mais robustos, autônomos, inteligentes e seguros para os robôs.

A adoção de *softwares* de simulação robótica realista tem um papel extremamente benéfico ao acesso à robótica. Pois torna possível recriar virtualmente um determinado robô (*hardware*) em um mundo virtual, e assim, desenvolver e preparar o software reduzindo custos, tempos de desenvolvimento e experimentação, ou riscos de dano a um robô real.

É de grande importância na atualidade à formação de pessoas capacitadas nesta área, com conhecimentos sobre o tema, e capacitadas para o projeto e desenvolvimento de aplicações robóticas. Neste curso serão abordados os conceitos sobre sensores, atuadores, configuração de robôs (modelos cinematográficos), e sobre a programação de comportamentos inteligentes, ou seja, a implementação de comportamentos integrando a percepção-decisão-ação do robô. O simulador adotado neste curso, o V-REP [E. Rohmer 2013]¹, permite a reprodução de situações reais com um alto grau de realismo (3D, simulação física), além de permitir a realização de testes com os mais variados modelos de robôs (por exemplo Pioneer, NAO Humanóide, Braços Robóticos, Robôs articulados com patas, robôs aéreos) e de sensores (Laser Sick e Hokuyo, Câmeras, GPS, Kinect, Velodyne). Este simulador é de uso livre para fins acadêmicos, permitindo uma fácil e ampla configuração e programação dos robôs. O uso de um simulador realista e flexível facilita assim um contato direto com a robótica e com as inúmeras possibilidades de desenvolvimentos de estudos e pesquisas nesta área².

6.2. Conceitos sobre Robôs Móveis e Robôs Articulados

Robôs podem possuir dois tipos de bases distintas. Os robôs de base fixa são aqueles que permanecem fixados a um local específico, como por exemplo, os braços robóticos industriais, e possuem articulações (por isso são denominados de Robôs Articulados ou Braços Manipuladores) que permitem efetuar o trabalho para o qual foram projetados. Já os de base móvel possuem a capacidade de se locomover pelo ambiente em que se encontram, esses robôs são chamados de Robôs Móveis por essa razão. Os robôs móveis podem ainda ser tele-operados, semi-autônomos, autônomos ou dotados de sistemas inteligentes de tomada de decisão, constituindo os denominados veículos autônomos inteligentes [Jung et al. 2005]. Portanto, estes últimos possuem a capacidade de locomoção e de operação semi ou completamente autônoma, atuando em ambientes totalmente controlados (ou não) e bem estruturados (ou não).

Os robôs são usualmente compostos por diversos dispositivos e módulos, destacando-se os sensores (capacidade de percepção do ambiente onde ele atua), o sistema de processamento de informações (planejamento, decisão e controle), e os atuadores (geram as ações, através de motores capazes de produzir estas ações). Nas próximas subseções

¹V-Rep Site: <http://www.coppeliarobotics.com/>

²Site com informações complementares e demos do V-REP: <https://www.sites.google.com/site/vrepjai/>

são mostrados alguns exemplos dos diversos sensores (Seção 6.2.1) disponíveis; em seguida os elementos usados para dar mobilidade e movimento as articulações dos robôs, os atuadores (Seção 6.2.2); na Seção 6.2.3 serão discutidos os diferentes tipos de atuadores e como estes podem afetar o comportamento e movimentação do robô de acordo com sua utilização; na sequência as arquiteturas de controle (Seção 6.2.4), ou seja, os modelos computacionais adotados para implementar o sistema de planejamento, decisão e ação de robôs são mostrados; e então na Seção 6.2.5 é apresentada uma discussão sobre os comportamentos mais simples adotados em sistemas robóticos móveis/articulados. Estes comportamentos é que nos permitem implementar tarefas robóticas mais sofisticadas (p.ex. mapeamento do ambiente, determinação da localização do robô em mapas, navegação até uma determinada posição, desvio de obstáculos), envolvendo a percepção, decisão e ação dos robôs.

6.2.1. Sensores

Sensores podem ser chamados também de transdutores já que a ideia por de trás deste equipamento é transformar a energia medida em outro formato de apresentação, mais simples para a utilização [Murphy 2000]. Por exemplo, um sensor pode captar o som, a pressão ou mesmo a luz, convertendo em um sinal analógico ou digital que possa ser usado pelo robô.

Existem diversos tipos de sensores [Wolf et al. 2009], porém podemos classificar em dois tipos principais segundo seu método de medição: os sensores ativos emitem energia no ambiente e mensuram seu retorno; e os sensores passivos somente captam a energia já disponível no ambiente [Siegwart et al. 2011].

A tarefa do robô deve ser levada em conta no momento da escolha do sensor, já que cada sensor possui características próprias, como por exemplo, faixa de atuação, sensibilidade, precisão e exatidão [Romero et al. 2014b]. Quando não existem disponíveis sensores com características compatíveis com a tarefa, uma alternativa adotada é a fusão de múltiplos sensores. A fusão sensorial pode ser classificada em [Romero et al. 2014b]: (i) redundante ou competitiva, onde, mais de um sensor com características iguais ou distintas fazem a medições do ambiente para posterior confronto (normalmente utilizada quando o problema a ser tratado é a imprecisão); (ii) complementar, sensores que medem informações distintas e que juntos são capazes de eliminar falsas detecções; e (iii) coordenada na qual se utiliza sensores em uma ordem específica, como por exemplo, sendo detectado som em alguma sala, então outro tipo de sensor é utilizado para se confirmar mais informações, como uma câmera.

Os sonares são sensores ativos capazes de detectar a distância entre o sensor e os objetos/obstáculos. A detecção ocorre pela emissão de um pacote sonoro ultrassônico, onde, mede-se o tempo em que o som demora a refletir em alguma superfície e retornar. Os sonares são equipamentos usualmente de baixo custo e possuem um alcance curto, detectando normalmente objetos entre 30 centímetros e 5 metros [Romero et al. 2014b]. A precisão da medição também depende do ângulo em que as ondas atingem uma superfície, quanto mais agudo maior é o erro da medição. Um robô pode utilizar vários sonares, porém precisam ser acionados de modo sequencial para minimizar eventuais interferências entre eles. Na Figura 6.2(a), mostra-se um exemplo de sonar o HC-SR04 da Cytron

Technologies.

Os sensores a *laser* são denominados de LIDAR (*Light Detection And Ranging*) e detectam distâncias de maneira similar ao cálculo feito pelo sonar, mas utilizando o tempo entre a emissão e recepção de pulsos de *laser*. A luz possui velocidade de propagação mais rápida que o som e melhor direcionada (no caso do laser), por isso os sensores LIDARs podem ter um ciclo de execução mais curtos e melhor precisão na estimativa de distâncias. Dois exemplos de sensores LIDAR utilizados em robótica são o sensor Hokuyo URG-04LX-UG01 [Hokuyo Automatic 2015] (Figura 6.2(b)) e o Sick LMS 200 [SICK 2015] (Figura 6.2(c)). O Hokuyo URG-04 é um sensor possui um alcance de 4 metros e um campo de visão de 240°. O Sick LMS-200 possui alcance de 10 metros (em condições ideais chega a 80 metros), um campo de visão de 180° e um custo significativamente maior que o Hokuyo para aquisição. Ambos os sensores redirecionam o feixe de *laser* usando um espelho³, permitindo assim, a varredura do ambiente. Existem alguns fatores que podem dificultar a medição das distâncias [Romero et al. 2014b]: objetos escuros (a cor preta absorve a luz/*laser*), quanto mais distantes os objetos mais impreciso é a leitura (menos luz retorna ao sensor) e a incidência do *laser* sobre superfícies com propriedades específicas de reflexão/refração/absorção de luz (ex: espelhos e vidros) causam falhas de leitura.

Câmeras de vídeo são sensores comumente aplicados em projetos robóticos, apresentando uma rica quantidade de informações sobre a área de atuação visualizada pelo robô [Romero et al. 2014b] [Mataric 2014]. Com este ganho de informação é possível dotar o robô de um sistema de percepção similar ao humano, obtendo distâncias (como sensores laser e ultrassom), presença, postura, gestos, etc. As câmeras atuais capturam a informação utilizando a tecnologia CCD ou CMOS. O CCD (*charged coupled device*) possui uma matriz sensível à luz (fotodíodo) ou *pixels*. A energia projetada sobre o *pixel* é acumulada por certo tempo, sendo então congelada e transmitida a um registrador. O registrador processa uma linha da matriz de pixels por vez, amplificando o sinal de cada *pixel* e convertendo para digital, gerando assim a imagem final. A Figura 6.2(d) apresenta a câmera Point Grey Chameleon[Point Grey Research 2015] que utiliza matriz CCD. A tecnologia CMOS (*Complementary Metal-Oxide Semiconductor*) possui uma matriz de pixel similar ao CCD, mas tendo como grande diferença, a existência em cada *pixel*, de um registrador dedicado, que amplifica e digitaliza o sinal capturado. Não necessitando, portanto, de congelamento ou transmissão de informações para um registrador externo. As duas tecnologias vem competindo pela preferência dos fabricantes de câmeras a alguns anos, mas o que deve ser levado em conta no momento de se optar por uma ou por outra é que a CCD possui maior sensibilidade a luz e menor apresentação de ruídos; já o CMOS pode trabalhar com uma frequência de captura mais elevada, possui menor custo de fabricação e menor consumo de energia [Siegwart et al. 2011]. CCD, portanto, possui uma qualidade de imagem melhor, mas CMOS vem apresentando significativas melhorias e já é possível encontrar câmeras CMOS com qualidade similar a CCD [Romero et al. 2014b].

As câmeras de vídeo permitem a aquisição de imagens, que são mais ricas em

³LIDAR - Animação do funcionamento: <http://upload.wikimedia.org/wikipedia/commons/c/c0/LIDAR-scanned-SICK-LMS-animation.gif>

termos de informações, porém usualmente mais complexas de serem tratadas. Os sistemas de visão computacional buscam explorar as informações capturadas pelas câmeras e assim contribuir na percepção do ambiente, seja usando imagens estáticas ou através de sequências de imagens (vídeos). As câmeras permitem identificar elementos pela sua forma e contornos, cor, textura e até mesmo estimar a proximidade de obstáculos.

Um sensor de posicionamento muito usado na robótica (para ambientes externos) é o *Global Positioning System* (GPS) ou Sistema de Posicionamento Global. O GPS utiliza-se dos satélites de posicionamento que estão na órbita da terra. Esses satélites emitem mensagens contendo a hora de partida da mensagem. Assim, o GPS calcula a distância que a mensagem percorreu para chegar e, tendo recebido mensagens de pelo menos 2 satélites [Siegwart et al. 2011], é possível estimar a posição do sensor na superfície da terra. O GPS é um sensor absoluto, ou seja, cada cálculo de posicionamento é independente do anterior, portanto, seu erro é local e não cumulativo [Romero et al. 2014b].

O giroscópio é um dispositivo capaz de informar a direção para onde está se movendo. Isso faz com que seja usado para auxiliar em navegação de helicópteros e aviões (principalmente no piloto automático). Existem dois tipos de giroscópios, os mecânicos e os ópticos [Siegwart et al. 2011]. Os mecânicos são baseados no princípio da inércia, usando um rotor suspenso por um suporte formado por dois círculos articulados (um exemplo de giroscópio mecânico é mostrado na Figura 6.2(e)). Os ópticos medem a velocidade angular entre dois feixes de luz ou *lasers* emitidos de uma mesma fonte. O erro de medição do giroscópio é cumulativo, ou seja, erros anteriores (ou perdas de leituras de dados) atrapalham nas medições futuras e estimativas de posicionamento e orientação.

Um acelerômetro também é uma unidade inercial como o giroscópio, porém, mede a ação de forças externas sobre ele, inclusive a gravidade. Atualmente, os acelerômetros são pequenos sistemas Microeletromecânicos (MEMS) que através de uma pequena coluna (massa) balançante, as forças são mensuradas. Um exemplo de acelerômetro é o MMA7361L [Freescale Semiconductor 2015], mostrado na Figura 6.2(f). Os acelerômetros, assim como o giroscópio, também sofrem do problema do erro cumulativo, que pode levar a erros em estimativas de posição e orientação. Por isso muitas vezes são aplicados filtros para eliminar ruídos e melhorar as estimativas, como o Filtro de Kalman [Wolf et al. 2009].

Existem dispositivos que agregam um giroscópio e um acelerômetro internos, podendo até fazer fusão de mais sensores, como bússola. Assim são capazes de estimar a posição relativa, velocidade e aceleração do movimento de um veículo ou robô. Esses dispositivos integrados são chamados de IMUs (*Inertial Measurement Units*), porém carregam os mesmos erros cumulativos do giroscópio e acelerômetro. É possível, no entanto, adotar alguma referência externa como um GPS para minimizar esse problema, bem como aplicar filtros. Um exemplo de IMU é o MPU6050 [InvenSense 2015] mostrado na Figura 6.2(g).

A profundidade do ambiente pode ser observada através de sensores 3D, como por exemplo, a câmera estéreo. Ao se utilizar algoritmos de processamento das imagens proveniente de duas câmeras defasadas (dois pontos de vistas diferentes, normalmente próximos) podem ser gerados mapas de disparidade [Bradski and Kaehler 2008], onde

a relação entre o mesmo pronto visto de diferentes posições (por cada uma das câmeras) permite estimar a profundidade dos elementos da cena, obtendo assim um mapa de profundidade (*depth map*). A câmera estéreo pode ser um sensor RGB-D (provendo cor RGB + *Depth*), ou seja, para cada *pixel* da imagem colorida adquirida (RGB) é apresentada também a sua respectiva profundidade na cena (distância para o sensor).

Outro importante sensor 3D que vem sendo muito adotado na robótica é o Kinect. Este sensor foi desenvolvido inicialmente para o *videogame* XBox 360 da Microsoft, mas posteriormente foram desenvolvidos drivers que permitem coletar seus dados por meio de diversos sistemas operacionais. Através dele são obtidos dados RGB-D da cena. O Kinect calcula a profundidade utilizando um emissor de um padrão quadriculado em NIR (*Near Infrared*) no ambiente e é capturada por um sensor NIR que processa a luz estruturada, obtendo assim a profundidade de cada *pixel* da imagem. A imagem RGB da cena é adquirida por uma câmera comum também presente no dispositivo, onde é feito o registro das imagens da câmera NIR com a câmera RGB. O Kinect, no entanto, foi desenvolvido para ser utilizado apenas em ambientes internos, onde não há a incidência direta da luz do sol, já que o sol emite luz infravermelha no mesmo comprimento de onda emitido/captado pelo seu sensor NIR, inviabilizando a obtenção da profundidade. Na Figura 6.2(h), o Kinect é mostrado.

Sensores 3D também podem ser baseados em laser multi-camadas (com múltiplos feixes), como é o caso do Velodyne. O Velodyne é um sensor que vem sendo utilizado em veículos autônomos, como o CaRINA [Fernandes et al. 2014] e o *Google Self-Driving Car*, para se detectar obstáculos no entorno do veículo, ou mesmo, para a construção de mapas. O Velodyne utiliza vários feixes lasers (podendo ser 32 ou 64 dependendo do modelo), que são rotacionados pelo dispositivo, alcançando 360 graus de campo de visão. Com base nas distâncias obtidas por cada um dos feixes é possível criar uma nuvem de pontos 3D que representa o ambiente. Os lasers do Velodyne podem medir distâncias a partir de 2 cm e podendo chegar a 120 metros em condições favoráveis [Siegwart et al. 2011]. O Velodyne HDL-32E [Velodyne LiDAR 2015] é mostrado na Figura 6.2(i).

Diversos sensores citados nesta seção estão disponíveis na sua forma simulada (bastante realista) junto ao simulador V-REP: Sonar, Lasers (Hokuyo URG-04 e Sick LMS-200, Velodyne), Câmeras, Kinect, GPS, Acelerômetro e Giroscópio. Além destes, outros sensores como os *bumpers* (sensores de contato) e de detecção de passagem também estão disponíveis no simulador.

6.2.2. Atuadores

Atuadores dotam os robôs da capacidade de produzir ações, ou seja, deslocar, manipular e interagir com o ambiente e seus elementos. Existem muitos tipos de atuadores robóticos (motores) disponíveis atualmente [Mataric 2014], dentre eles, pode-se citar: (i) os rotatórios - os motores do tipo servo que permitem um controle fino do posicionamento; motores de corrente contínua que são empregados no deslocamento baseados em rodas ou esteiras (avança e recua); e o motor de passo que é preciso no controle angular de giro; (ii) os lineares - motores que criam um movimento linear; (iii) os pneumáticos; e (iv) os hidráulicos.

Os atuadores serão dispostos de modo a garantir os movimentos das juntas dos



Figura 6.2. Exemplo de sensores: (a) sonar HC-SR04 [Cytron Technologies 2015], os sensores laser (b) URG-04LX-UG01 [Hokuyo Automatic 2015] e (c) LMS 200 [SICK 2015], (d) câmera Point Grey Chameleon [Point Grey Research 2015], (e) Giroscópio mecânico, (f) acelerômetro MMA7361L [Freescale Semiconductor 2015], (g) IMU MPU6050 [InvenSense 2015], (h) Microsoft Kinect e (h) Velodyne HDL-32E [Velodyne LiDAR 2015].

robôs, sejam estes movimentos circulares contínuos (p.ex. em rodas), angulares (p.ex. em juntas de braços e pernas) ou lineares (p.ex. pistão). Os atuadores irão definir os graus de liberdade de movimento do robôs, sendo adotados de acordo com a configuração e a tarefas específicas definida para o robô, bem como possuindo uma precisão e desempenhos (velocidade, aceleração, torque, capacidade de carga) próprios.

No caso dos robôs móveis, a forma como estes atuadores são usados para controlar o seu deslocamento irá definir a sua movimentação (cinemática) e a aplicação de movimento aos motores irá resultar em uma trajetória própria de acordo com esta configu-

ração, como por exemplo, em um veículo que utiliza uma tração única na traseira (atuador das rodas traseiras) e direcionamento por uma barra de direção frontal (atuador de giro da direção, ou, de esterçamento).

6.2.3. Cinemática e Dinâmica de Robôs Móveis

A cinemática e a dinâmica podem ser estudadas separadamente, mas é interessante, quando se objetiva a análise do comportamento do robô, seu estudo em conjunto [Vieira and Roqueiro 2014].

A cinemática estuda o movimento, velocidade e acelerações de robôs móveis e articulados, assim, torna-se possível descrever a trajetória necessária para o deslocamento de um robô de uma pose (posição e orientação) inicial até a pose final. Nos robôs móveis com rodas, é usual a adoção de modelos com cinemática diferencial (controle independente de 2 rodas), permitindo que o robô gire ao redor de sua própria base (Figura 6.1(c)) e Figura 6.1(e)), por outro lado outro modelo cinemático bastante adotado é o dos veículos com barra de direção (com aceleração, freio e esterçamento), denominado de cinemática do tipo Ackermann [Dudek and Jenkin 2000]. O deslocamento de robôs com patas e pernas permite criar uma movimentação própria que se diferencia bastante da cinemática dos robôs móveis com rodas, assim como os veículos baseados em propulsão (aquáticos e aéreos).

A dinâmica também se preocupa com o movimento, velocidade e aceleração do robô, mas vai além, seu intuito está em estudar as forças e momentos envolvidos, incluindo atrito, gravidade, colisões e a reação às colisões. As equações de dinâmica do movimento são bases para inúmeros algoritmos computacionais de projeto mecânico, controle e simulação.

Tanto a cinemática como a dinâmica levam em conta as especificidades de cada robô, ou seja, a forma como se locomovem (rodas, esteiras, esferas, pernas ou pistões) e os tipos de atuadores empregados na efetivação dos movimentos. O simulador V-REP permite que sejam simulados os modelos tanto cinemático quanto dinâmico (física dos movimentos) em robôs móveis e articulados, podendo simular robôs com cinemática diferencial, Ackermann, robôs com esteiras, pernas e patas, e inclusive robôs aéreos. Uma vez que o modelo de simulação adotado respeita a cinemática e dinâmica dos robôs, isto significa que um robô pode cair, desequilibrar-se, onde podemos inclusive simular situações como a de um robô batendo em um obstáculo que pode se mover (ser empurrado) ou pode ser estático (bloco pesado de concreto).

6.2.4. Arquiteturas de Controle

Na robótica móvel, a arquitetura é a maneira pela qual se constrói um software que controla de maneira inteligente um robô [Grassi Jr. and Okamoto Jr. 2014]. A arquitetura apresenta portanto os módulos necessários para o funcionamento do sistema e de que maneira esses módulos interagem.

Há três grupos principais de módulos e componentes [Iyengar and Elfes 1991]. O grupo da Percepção envolve atividades de interpretação de sensores, modelagem do ambiente e reconhecimento. Já o grupo de Planejamento é responsável por planejar as tarefas, sincronizar e monitorar o funcionamento do robô. Por fim, o grupo da Atuação

executa os movimentos e ações propriamente ditos, controlando portanto, os atuadores.

O ciclo de tarefas composto por esses três componentes é mostrado na Figura 6.3. Na etapa de percepção o robô faz a detecção dos obstáculos por meio de seus sensores, em seguida mapeia os obstáculos criando um modelo interno para o ambiente e, por fim, atualiza sua própria posição (localização). Com base nessas informações, passa a buscar uma trajetória livre de colisão para chegar ao seu destino, na etapa de planejamento. E então, ações são tomadas por meios dos atuadores (ativação de motores), fazendo com que o robô avance, recue ou efetue curvas no momento planejado. Sendo o ambiente dinâmico, essa sequência de tarefas deve ser repetida indefinidamente.

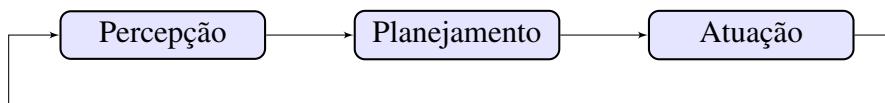


Figura 6.3. Ciclo percepção-planejamento-atuação [Grassi Jr. and Okamoto Jr. 2014].

A arquitetura de um robô pode ser classificada em: deliberativa, reativa e híbrida (combinando a deliberativa e a reativa) [Arkin and Balch 1997, Murphy 2000]. A deliberativa utiliza um modelo interno do mundo para planejar antecipadamente (deliberando) as ações do robô a fim de atingir seu objetivo. A arquitetura deliberativa se baseia no contexto e em conhecimentos sobre o mundo e a situação em que se encontra o robô. A arquitetura reativa toma ações em resposta imediata a uma percepção (reação sensorial-motora), com reações predefinidas ao se deparar com uma informação sensorial local (por exemplo, um obstáculo). Uma arquitetura híbrida faz uso de conhecimentos de mais alto nível (p.ex. mapas e informações do nível deliberativo) para planejar suas ações, mas pode ter também a capacidade de reagir a situações inesperadas e percepções do estado atual do ambiente (p.ex. reagindo a estas percepções e elementos dinâmicos do ambiente).

Portanto, as arquiteturas reativas são fortemente indicadas a ambientes dinâmicos, as arquiteturas deliberativas são mais gerais e flexíveis na definição de ações [Arkin 1989, Mataric 1992] (precisam de pouca adaptação para funcionar em ambientes já modelados), e as arquiteturas híbridas visam extrair o melhor da deliberação e reação para a tarefa afim. Na Figura 6.4, é mostrado um diagrama entre as classificações de arquiteturas.

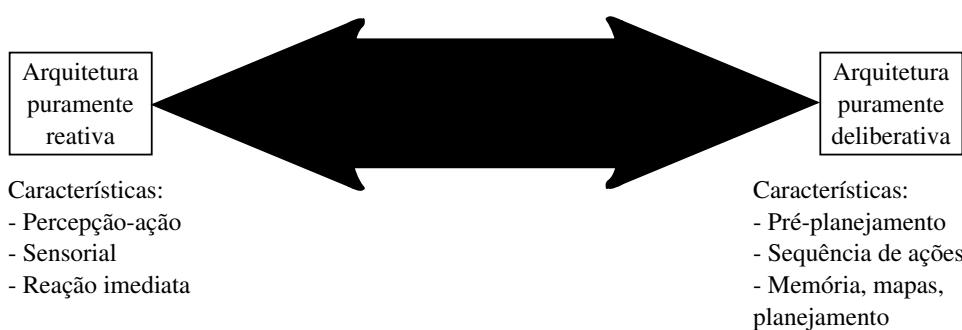


Figura 6.4. Classificação de arquiteturas baseada em deliberação e reatividade [Grassi Jr. and Okamoto Jr. 2014].

6.2.5. Comportamentos

Um robô pode ter diferentes tipos de comportamentos, onde inicialmente é necessário definir o seu grau de autonomia. Os robôs mais simples são aqueles que apenas executam de forma repetitiva ações sem considerar informações adicionais, os autômatos, seguidos pelos tele-operados, onde toda a percepção e decisão é humana e apenas a ação é executada pelo robô. Os robôs mais sofisticados, e que são o foco mais central deste texto, são os robôs que possuem um certo grau de autonomia, ou seja, possuem a capacidade de perceber o ambiente e agir de acordo com esta sua percepção e planos para executar uma determinada tarefa Figura 6.5.

	Percepção	Decisão	Ação
	Sensores	Processamento	Motores
Autômato	-	-	R
Tele-Operado	H	H	R
Semi-Autônomo	H/R	H/R	R
Autônomo	R	R	R

H: Humano – R: Robótico

Figura 6.5. Níveis de autonomia de robôs móveis e articulados

Quando se pensa em projetos de robôs autônomos, a arquitetura mais simples que pode ser adotada são os comportamentos reativos. Por meio de alguns sistemas simples e alguma preparação do ambiente é possível fazer com o robô realize tarefas importantes para empresas ou para o cotidiano das pessoas.

Uma dos comportamentos reativos muito utilizados é o do “seguidor de linha” (*Line-following behavior*). Consiste em fazer com que o robô tenha a capacidade de seguir uma simples linha no chão, e faz assim com que ele consiga na verdade seguir rotas pré-definidas, mas sem a necessidade de conhecer o mapa completo do ambiente e nem a sua localização precisa. Assim pode-se realizar uma série de operações como a de carregar mercadorias dentro de um almoxarifado ou até mesmo de um galpão de enormes proporções como o da gigante Amazon. O estoque de mercadorias da Amazon é controlado por robôs seguidores de linhas da Kiva Systems, e esta dependência destes robôs levou inclusive a gigante do varejo a adquirir a fabricante destes em 2012 [Guizzo 2012]. A Figura 6.6 mostra os robôs seguidores de linhas em operação.

O Roomba (ver Figura 6.1(e)) da iRobot [iRobot 2015] é outro exemplo de robô reativo, ele é capaz de aspirar o pó de um local utilizando rotas quase totalmente aleatórias (*Wander behavior*) apenas vagando pelo ambiente e evitando de colidir com os obstáculos por meio de seu sensoriamento, e fazendo com que busque cobrir todas as partes de uma sala pelo menos uma vez (e sem conhecer previamente o local!). O Roomba e praticamente todos os robôs baseados em comportamentos reativos possuem detecção de obstáculos, assim ele são capazes de se adaptar ao mundo dinâmico ao seu redor, sem colidir contra os elementos do ambiente (estáticos ou dinâmicos).



Figura 6.6. Robôs seguidores de linha que efetuam o transporte de mercadorias na Amazon [Guizzo 2012].

Seguir paredes (*Wall-following behavior*) também é uma estratégia reativa inteligente, já que com esta simples ideia é possível percorrer todo um andar de um edifício. Essa estratégia pode ser empregada em robôs na geração de um mapa do ambiente em que está inserido e, posteriormente, é possível navegar de maneira deliberativa usando o mapa assim construído.

Outro comportamento interessante é o de acompanhar uma pessoa (*Follow-me behavior*) [Correa 2013]. Há alguns robôs que precisam interagir com pessoas, para esses robôs é muito importante que tenham a capacidade de acompanhar uma pessoa ao caminhar e se deslocar pelo ambiente. Um robô também pode seguir um outro robô/veículo (autônomo ou não) ou outros elementos móveis presentes no ambiente.

Inclusive podemos combinar comportamentos reativos (por fusão ou por seleção/arbitramento [Mataric 2014]) a fim de criar comportamentos mais complexos, como por exemplo, combinar um comportamento de navegação em direção a um alvo (p.ex. usando uma bússola ou uma coordenada GPS) e um comportamento de desvio de obstáculos.

Um exemplo interessante de um projeto avançado de robôs móveis inteligentes é o do projeto CaRINA [Fernandes et al. 2014], que é uma plataforma robótica utilizada no desenvolvimento de sistemas de percepção, controle e tomada de decisão para navegação autônoma e assistiva de veículos em ambientes urbanos. Ele conta com um controle computacional de esterçamento, aceleração e frenagem, e diversos sensores como: GPS, IMU, câmeras e lasers. A primeira versão do sistema autônomo, do CaRINA I (veículo elétrico), funcionava com um sistema de visão capaz de classificar imagens em "zona navegável"(asfalto) e "não navegável"(calçada, obstáculos), podendo ser orientado por coordenadas de GPS. De certo modo, podemos até dizer que este era um "seguir de linhas"(asfalto) com direcionamento por GPS. Este sistema permitiu uma navegação por mais de 1Km no Campus da USP São Carlos usando apenas 7 pontos de coordenadas GPS como referência e se mantendo "dentro da linha da rua", tendo este experimento sido realizado em Outubro de 2011.

Certamente que para os robôs poderem desempenhar tarefas em ambientes mais

complexos e dinâmicos é interessante e importante a presença de comportamentos deliberativos e reativos (híbrido) em conjunto. Um exemplo desse tipo de projeto mais complexo é o CaRINA II (carro de passeio automatizado e autônomo) [Fernandes et al. 2014], o qual é uma plataforma robótica que possui a capacidade de navegação autônoma em ambiente urbano, tendo sido o primeiro veículo completamente autônomo a navegar em ruas urbanas, misturado ao tráfego local, no Brasil. O CaRINA 2 possui uma gama de sensores como GPS, IMU, câmeras e *lasers*, que são mostrados na Figura 6.7, que facilitam a sua percepção do mundo. Através dos dados provenientes desses sensores, alguns sistemas percebem o ambiente exterior de maneira dinâmica, como: detecção de obstáculos, detecção de pista de rodagem, detecção de cruzamentos, detecção de meio-fio, detecção de tráfego e detecção da região navegável pelo veículo. As rotas do CaRINA 2 são geradas por meio de um *waypoint* de coordenadas GPS, onde são usados mapas adaptados de sua trajetória, levando-se em conta a sua localização e as informações aferidas do ambiente pelos sistemas de percepção. O CaRINA 2 usa o sistema ROS (Robot Operating System) para a sua operação e possui uma versão de simulação realística do veículo usada para testes.

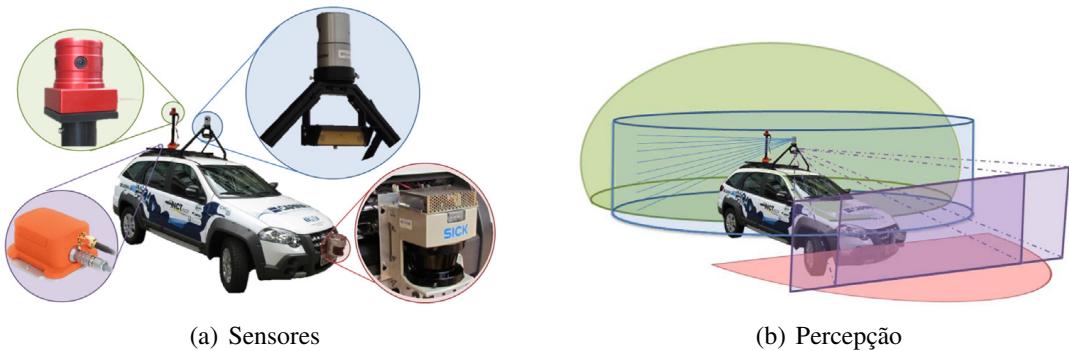


Figura 6.7. Carro Robótico Inteligente para Navegação Autônoma - CaRINA 2 - visão sensorial [Fernandes et al. 2014].

6.3. Simulação de Robôs

O uso de simuladores robóticos vem crescendo muito nos últimos anos. A simulação vem sendo usada para validar e contribuir para o desenvolvimento de algoritmos robóticos. Os avanços obtido no desenvolvimento de jogos eletrônicos vem sendo incorporado à simulação nos últimos anos, permitindo um maior realismo gráfico e nos cálculos de simulação física envolvidos, resultando em uma alta qualidade de simulação da cena [Harris and Conrad 2011]. Atualmente os simuladores já tem capacidade de simular praticamente qualquer tipo de robô, sensores e ambientes. Alguns simuladores tem se destacado no cenário da robótica (Player/Stage, Gazebo, Morse, V-REP, Webots, MRS-Microsoft Robotics Simulator, Matlab Robotics Toolbox), onde as principais ferramentas livres para uso acadêmico serão listadas a seguir.

6.3.1. Player / Stage / Gazebo

O Projeto Player foi iniciado em 1999 por pesquisadores da University of Southern California, e posteriormente foram agregados pesquisadores de diversas instituições, sendo

um dos ambientes mais amplamente usados em pesquisa. O Player/Stage/Gazebo tiveram uma importante contribuição junto ao ROS (Robot Operating System). O projeto é código aberto e de distribuição livre (*Open Source*), e dessa forma, está constantemente sendo desenvolvido e adequado para um número cada vez maior de robôs e sensores comercialmente acessíveis. O Player está estruturado em uma arquitetura do tipo cliente e servidor, ou seja, baseado em trocas de mensagens por uma rede de computadores (TCP/IP). A interface com o robô e seus sensores é feita por meio do servidor. O servidor obtém os dados (robô e sensores), disponibilizando-os para o cliente e, ainda, recebe instruções do cliente e as transmite ao robô. Usualmente o programa que controla o robô é o cliente (controlador "inteligente"), sendo responsável por receber os dados do servidor, processá-los e enviar as instruções para o servidor, que interfaceia e aciona o hardware do robô. O servidor funciona como uma camada de abstração, ou seja, por meio dele é possível que o mesmo cliente controle diferentes tipos de robôs, sem a necessidade de alteração de seu código, ou mesmo, que controle diversos robôs e sensores (servidores) ao mesmo tempo [Staranowicz and Mariottini 2011].

Existem muitas bibliotecas compatíveis com o cliente/servidor do Player, dentre elas pode-se citar as linguagens: C, C++, Java e Python. As bibliotecas do Player, além de possibilitar o controle do robô, possuem algoritmos de desvio de obstáculos, planejamento de trajetória e mapeamento de ambientes.

O Stage é um simulador robótico bidimensional voltado, principalmente, para ambientes internos. Permite a simulação de múltiplos robôs e sensores ao mesmo tempo, utilizando um ou mais clientes. Com o Stage pode-se simular em 2D e 1/2, ou seja, adicionar perspectiva na visualização e uma falsa impressão de 3D a simulação, porém toda simulação é feita em 2D (sensores, atuação). Na Figura 6.8, são mostrados os modos 2D e 2.5D do Stage. O Pioneer [Adept MobileRobots 2015] é o robô mais comumente adotado no Stage, inclusive por ele ser muito utilizado em laboratórios de pesquisa. A simulação do Stage abrange o deslocamento dos robôs e sensores como: odômetros, *lasers*, sonares, câmeras e garras. Os sensores no Stage se comunicam exatamente da mesma forma como seria feito com o hardware real, permitindo que o mesmo código testado em simulação seja utilizado na prática com um robô real [Vaughan et al. 2003]. Contudo, não existem garantias de que a simulação irá possuir uma fidelidade nos cálculos e comportamentos físicos [Gerkey et al. 2003].

Gazebo⁴ é outro simulador compatível com o Player [Koenig and Howard 2004], porém, cria um mundo 3D, ou seja, é mais realista e pode criar ambientes mais complexos que o Stage. Todas as funções utilizadas no Player/Stage podem ser utilizadas no Player/Gazebo sem qualquer modificação. O Gazebo utiliza as bibliotecas gráficas (OGRE) e de modelagem e simulação Física (ODE) que permitem uma fidelidade do comportamento e interações físicas entre robôs e objetos do ambiente [Craighead et al. 2008]. Todos os objetos da simulação possuem massa, fricção e uma série de atributos que permitem aos objetos um maior realismo ao serem empurrados, puxados, derrubados ou transportados [Staranowicz and Mariottini 2011]. O Gazebo deve ser empregado apenas quando a simulação 2D do Stage é insuficiente para o desenvolvimento dos experimentos, já que, o Stage é menos custoso computacionalmente. Portanto, seu uso torna-se inevi-

⁴Gazebo Simulator: <http://gazebosim.org/>

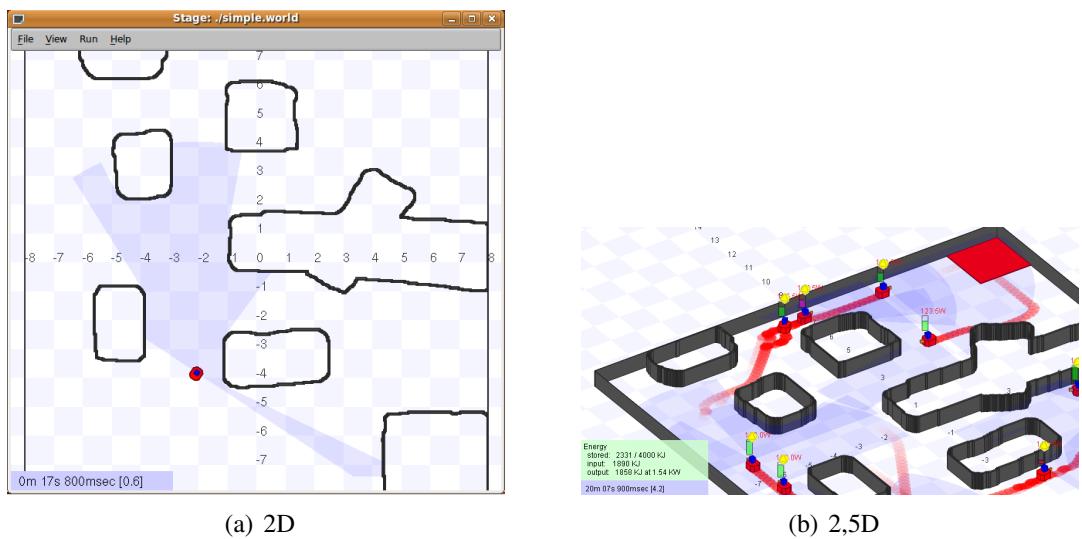


Figura 6.8. Exemplo de simulação Player/Stage, onde, (a) mostra o Stage em 2D e (b) o Stage em 2.5D. [Fonte: <http://playerstage.sourceforge.net/?src=stage>]

tável para ambientes externos (com solo irregular) ou quando a altura dos objetos ou a posição e orientação dos sensores deve ser levada em consideração na simulação do robô. A Figura 6.9 mostra um exemplo da simulação Player/Gazebo.

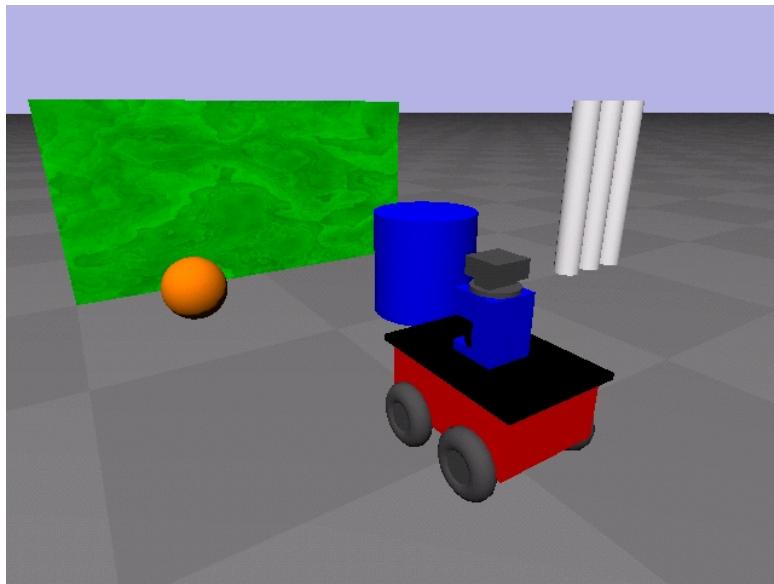


Figura 6.9. Exemplo de simulação Player/Gazebo. [Fonte: <http://playerstage.sourceforge.net/gazebo/gazebo.html>]

6.3.2. ROS

ROS (Robot Operating System)⁵ é atualmente uma das mais populares ferramentas da robótica, possuindo licença BSD (código aberto) [Quigley et al. 2009]. Seu suporte

⁵ROS - Robot Operating System: <http://www.ros.org/>

abrange diversos tipos de robôs e sensores, funcionando com baseado em serviços (*publish/subscribe*). O ROS possui uma comunidade muito ativa, disponibilizando uma série de pacotes e códigos prontos, o que facilita o seu desenvolvimento. O ROS é basicamente um "sistema operacional robótico" feito para permitir o acesso as diversos equipamentos e dispositivos robóticos (sensores, atuadores) e provendo também inúmeras bibliotecas de módulos avançados para o controle dos robôs. Portanto, o ROS é usado principalmente com robôs reais.

Por outro lado, para a simulação é possível usar o ROS em conjunto com alguns simuladores como, por exemplo, o Gazebo (ver Seção 6.3.1), incluindo as funções de mais alto nível disponíveis para o próprio ROS e Player. Deste modo, o ROS passa a controlar os robôs, sensores e atuadores simulados pelo Gazebo.

6.3.3. MORSE / Blender

MORSE (*Modular OpenRobots Simulation Engine*⁶) [Echeverria et al. 2011] é uma proposta acadêmica de simulador robótico, com desenvolvimento iniciado pelo *Laboratoire d'Analyse et d'Architecture des Systèmes* da Universidade de Toulouse, França. ROS (ver Seção 6.3.2), YARP (Yet Another Robot Platform) [Metta et al. 2006], Pocolibs [Pocolibs 2015], e MOOS [Oxford Mobile Robotics 2015] são plataformas robóticas que podem ser utilizadas em conjunto com a simulação do MORSE, permitindo assim o controle de robôs móveis em ambientes Terrestres, Aquáticos e Aéreos. É possível utilizar uma gama de sensores na simulação, dentre eles: acelerômetros, *lasers*, câmeras, e câmeras que capturem a profundidade. Com o MORSE, pode-se ainda criar facilmente componentes customizados como por exemplo, sensores e atuadores, que não acompanhem o pacote de instalação convencional. Toda a programação e customizações do MORSE podem ser feitas por meio de *scripts* em Python. As visualizações e simulações dos ambientes 3D são baseadas no Blender (OpenGL) e a fidelidade física é garantida pelo emprego do motor Bullet [Cook et al. 2014] (engine física). Um exemplo de simulação no MORSE é mostrada na Figura 6.10.



Figura 6.10. Exemplo de simulação no MORSE. [Fonte: <https://www.youtube.com/watch?v=kGx5SzQ3YWQ>]

⁶MORSE: <http://www.openrobots.org/morse/>

6.3.4. V-Rep

O V-REP⁷ é um simulador robótico 3D desenvolvido pela Coppelia Robotics. O simulador possui dupla licença, para utilização comercial é necessária à aquisição de licença e para uso educacional é gratuito (Free Educational e GNU GPL⁸). Ele suporta múltiplos motores de simulação física (Engines ODE, Bullet, Vortex), permitindo inclusive a troca durante a simulação (em tempo real). Além de prover a simulação, o V-REP [E. Rohmer 2013] permite a integração do desenvolvimento dos controladores do robô e customização do ambiente utilizando a linguagem de script Lua. É possível ainda programar a simulação usando *plugins*, ROS e com APIs disponíveis em diversas linguagens de programação como C/C++, Java, Python e Matlab.

O V-REP possui suporte a diversos tipos de robôs móveis e articulados (p.ex.: Pioneer, E-Puck, Khepera, Kuka Youbot, NAO, Baxter e manipuladores ABB, Kuka e Adept), atuadores e diversos sensores (p.ex.: *lasers*, câmeras, Kinect, Velodyne, GPS, acelerômetros) sendo possível ainda o desenvolvimento de novos atuadores e sensores via *plugins*. O simulador suporta robôs terrestres, aquáticos e inclusive aéreos, porém, para utilização de robôs aquáticos é necessária à criação de um ambiente apropriado. Na Seção 6.4 são apresentados mais detalhes do V-REP, que foi o simulador escolhido para ser adotado neste curso. As principais vantagens do V-REP são: a sua simplicidade de uso, a flexibilidade de programação e interfaces, a enorme variedade de modelos de robôs e equipamentos simulados, a facilidade de edição das cenas para a simulação, a disponibilidade para diferentes plataformas de Hardware (Windows, Linux e Mac - sem requisitos especiais em termos de configuração de processador, memória e/ou placa gráfica), e por ser distribuído de forma gratuita e aberta ao desenvolvimento para o uso acadêmico/educacional.

6.4. Uso e Desenvolvimento de Aplicações no Simulador V-REP

O Simulador V-REP permite simular diferentes tipos de bases robóticas, que incluem robôs como o Pioneer P3DX [Adept MobileRobots 2015] com cinemática diferencial, robôs de cinemática Ackermann (tipo carro com barra de direção, aceleração e freio), robôs humanoides como o NAO [Aldebaran Robotics 2015], robôs articulados com patas, braços robóticos manipuladores entre outros modelos disponíveis. O V-REP possui ainda uma ampla gama de sensores *laser* (Hokuyo, Sick, Velodyne), câmeras, Kinect, GPS, etc. A Figura 6.11 mostra um exemplo de tela do simulador⁹, incluindo 3 robôs: NAO, Pioneer, Spider e um Kinect sobre uma mesa. Vários exemplos de cenas e simulações usando o V-REP relacionados a este curso podem ser encontrados na Internet¹⁰.

É possível com o V-REP modelar o ambiente e os robôs, onde podemos adicionar objetos, paredes e outros elementos no ambiente virtual 3D, assim como podemos também criar novos robôs adicionando componentes, como por exemplo, é possível adicionar facilmente um sensor *laser* Sick ou Hokuyo ao robô Pioneer apresentado na Figura 6.11. Na Seção 6.4.1 é apresentada a interface do V-REP com o usuário, mostrando suas prin-

⁷V-Rep: <http://www.coppeliarobotics.com/>

⁸V-Rep Licensing: <http://www.coppeliarobotics.com/licensing.html>

⁹Vídeo da simulação desta cena está disponível em https://youtu.be/V_4vFyGGEDg.

¹⁰V-REP Curso JAI: <https://www.sites.google.com/site/vrepjai/>

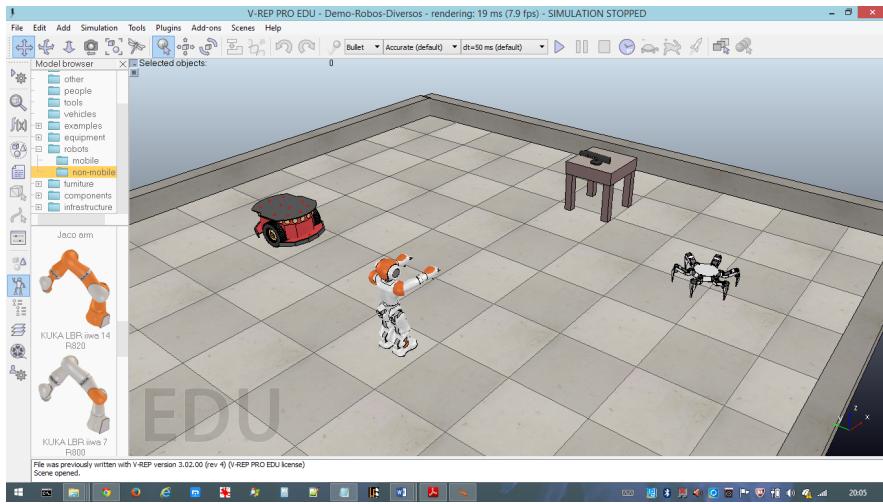


Figura 6.11. Simulador V-REP com ambiente 3D e Robôs Móveis/Articulados.

cipais funções e ferramentas. Particularidades sobre a cena e modelos empregados para a simulação são explanados na Seção 6.4.2. A Seção 6.4.3 apresenta algumas informações adicionais sobre o controle da simulação e quais são os parâmetros principais envolvidos. Além disto, é apresentada na Seção 6.4.4 os tipos e meios de se programar no V-REP, ou seja, como desenvolver programas que possam acessar os dados dos sensores, processar estes dados e enviar comandos para os motores do robô.

6.4.1. Interface com o usuário

A aplicação V-REP é composta por uma janela de console e uma janela de aplicação. Por meio da janela de console é executado o V-REP e são mostradas as impressões em tela do Lua, quais plugins estão sendo carregados e se foram corretamente inicializados, ou seja, é exibido todo o registro de informação relevante do sistema. Somente no Linux é obrigatória a execução do V-REP partindo-se de um terminal, porém, é uma prática recomendável em certas circunstâncias para os outros sistemas operacionais também.

A janela de aplicação é onde a cena é mostrada, editada, simulada e onde são feitas as interações com o sistema e a simulação. O usuário pode abrir muitas cenas em paralelo, mas somente uma pode estar ativa (em simulação ou edição). A Figura 6.12 mostra um exemplo da janela de aplicação e seus componentes que são descritos a seguir:

- *Componentes Hierárquicos na Cena:* é o componente que mostra a estrutura hierárquica de todos os objetos que compõem a cena. Cada objeto possui um ícone que identifica seu tipo, um nome e, em alguns casos, há um botão para a edição do *script* Lua que o rege e para a janela de parâmetros do *script* (Fig. 6.13). Clicando uma vez sobre o objeto ele é selecionado e destacado na cena. Ao clicar duas vezes sobre o ícone de um objeto a janela de propriedades do objeto é aberta. A habilitação da edição do nome do objeto se dá ao clicar duas vezes sobre seu nome. Objetos podem ser arrastados para dentro e fora de outros objetos, alterando assim o relacionamento entre os componentes da cena. O componente de hierarquia na cena pode não estar visível, já que há um botão na Barra de Ferramentas 1 permite

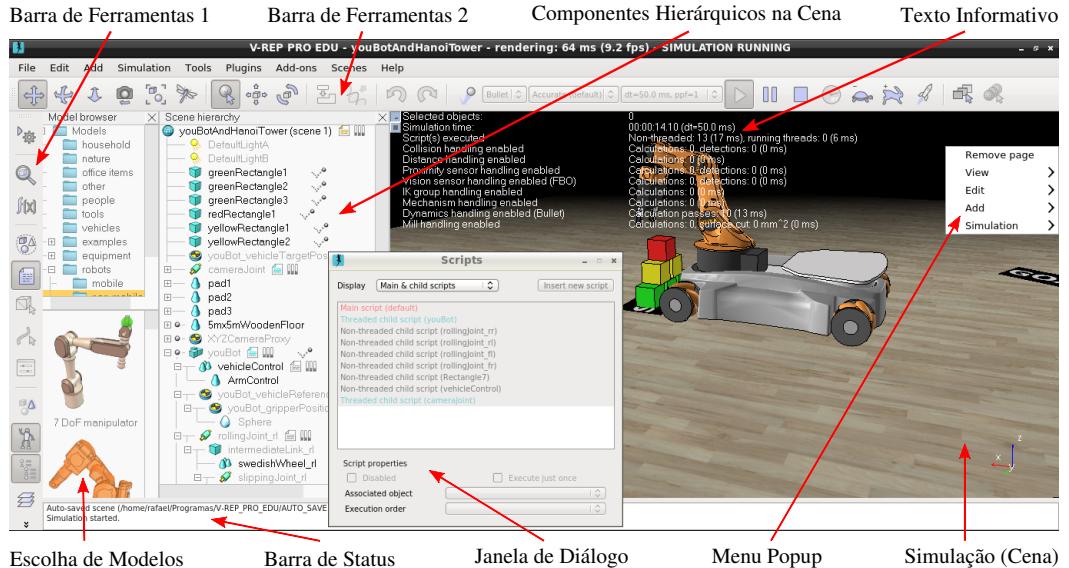


Figura 6.12. Componentes de tela do V-rep.

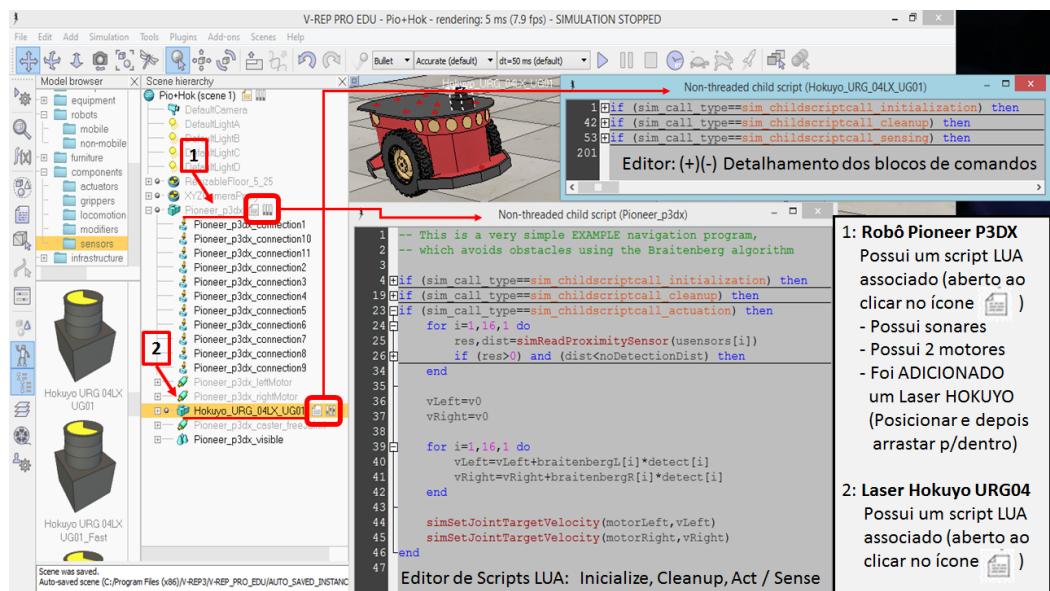


Figura 6.13. Robô e Sensores com Scripts LUA do V-rep.

que seja exibido e escondido.

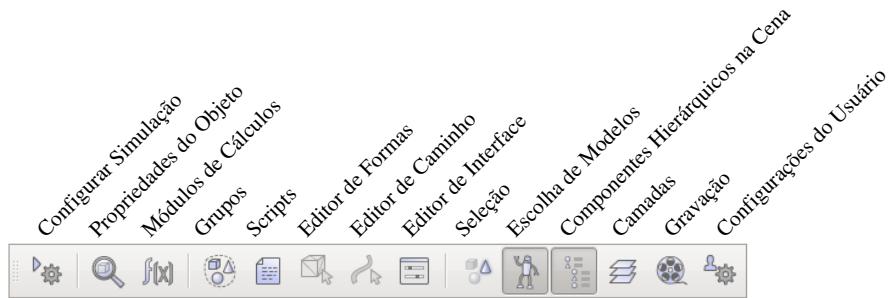
- **Texto Informativo**: exibe as informações referentes ao estado e parâmetros atuais do objeto/item selecionando na cena simulada.
- **Menu Popup**: é exibido ao clicar com o botão direito do mouse. Seu conteúdo depende do componente ao qual foi clicado e funciona como atalho rápido para funções dos objetos e componentes.
- **Janela de Diálogo**: Existem ainda várias janelas auxiliares, aos quais, permitem que

o usuário edite e interaja com a cena, possibilitando a alterações das configurações e parâmetros específicos de cada objeto ou item envolvido.

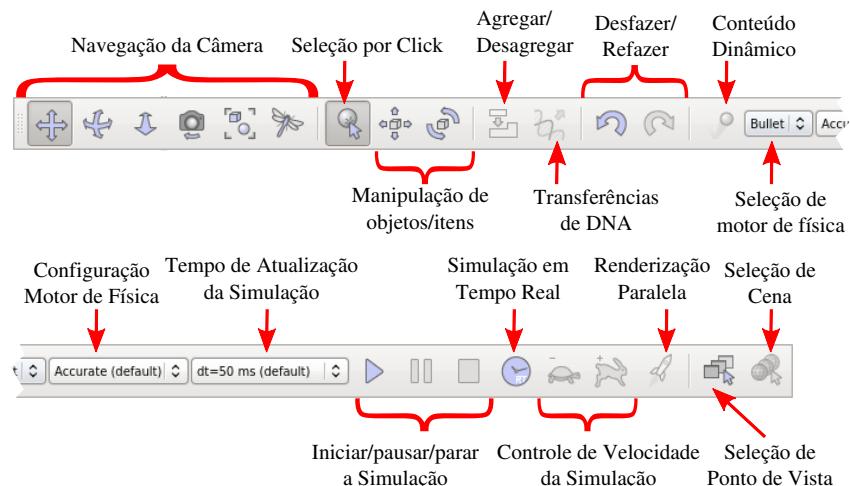
- *Barra de Status*: exibe informação relativa as operações e comandos, mostrando sempre as mensagens de erros provenientes do interpretador Lua. Pode-se enviar alguma informação para a Barra de Status usando a função simAddStatusbarMessage.
- *Escolha de Modelos*: exibe a estrutura de modelos disponíveis na instalação do V-REP. A estrutura é exibida através de uma árvore, ao qual, organiza os modelos conforme sua finalidade. Abaixo da árvore são mostradas as miniaturas de modelos pertencentes à pasta selecionada. Para incluir um modelo na cena, basta clicar sobre o modelo desejado e arrastar até a posição da cena desejada. Alguns modelos já trazem um *script* Lua padrão ao serem incluídos (Figura 6.13). O componente de escolha de modelos está visível por padrão, porém pode ser mostrado e escondido através de um botão na Barra de Ferramentas 1.
- *Barra de Ferramentas 1*: possui botões de atalho para a configuração da simulação, propriedades dos objetos, módulos de cálculo, os *Scripts* presentes na cena, edição de formas/caminho/interface, habilitação de seleção de objetos na cena, exibe/esconde o componente de Componentes Hierárquicos na Cena e Escolha de Modelos, Camadas, geração de vídeos da simulação e, por fim, as configurações globais do V-REP. A Figura 6.14(a) exibe os botões disponíveis na Barra de Ferramentas 1.
- *Barra de Ferramentas 2*: possui funções referentes ao controle da navegação da câmera (ponto de vista - camera pan, rotate e shift), habilita ou desabilita a seleção de objetos por clique, agrupa ou desagrega objetos selecionados, desfazer (undo) e refazer (redo) as alterações feitas, habilitar e desabilitar o uso de modelos dinâmicos na simulação (renderização e formas mais complexas), permite o deslocamento e rotação de objetos (object shift e rotate), escolha do motor responsável pela física (inclusive durante a simulação), escolha do tempo para atualização da simulação, iniciar/pausar/parar a simulação, habilitar a simulação em tempo real (procura manter a simulação sincronizada ao tempo real discorrido), alterar a velocidade em que a simulação ocorre (habilitada somente quando a simulação não for em tempo real), permitir que o V-REP realize a renderização em paralelo, seleção de ponto de vista e seleção da cena. Na Figura 6.14(b) é mostrada a Barra de Ferramentas 2.

6.4.2. Cenas e modelos

Os principais elementos das simulações são a cena e os modelos. Um modelo nada mais é do que um subelemento da cena, sendo possível incluir em uma cena vários modelos. Somente a cena pode ser simulada. Cenas são salvas com a extensão de arquivo “ttt”, onde este arquivo contém toda a informação necessária para a reabertura e simulação da cena. Já os modelos possuem a extensão “ttm” e possuem todas as informações necessária para a criação de um objeto de seu tipo. Um modelo pode ser criado a partir da combinação de outros modelos, podendo inclusive um objeto ser importados a partir de softwares



(a) Barra de ferramentas 1



(b) Barra de ferramentas 2

Figura 6.14. Barras de ferramentas

externos de modelagem 3D. A Figura 6.15 traz um exemplo de modelo de robô Pioneer P3DX inserido na cena, ao qual, a entidade criada na cena possui propriedades próprias, customizáveis e sem vinculação com o modelo gerador.

Os elementos presentes na cena e no modelo são similares, no entanto a cena possui alguns elementos a mais que são: o ambiente, um *script* principal e os pontos de vista da cena. Nas subseções a seguir são expostas informações adicionais sobre entidades e o ambiente no V-REP.

6.4.2.1. Ambiente

O ambiente no V-REP é definido por algumas propriedades e parâmetros que fazem parte da cena, mas não são objetos da cena. Alguns parâmetros que podem ser definidos no ambiente são: cores de fundo, parâmetros para inclusão de nevoeiro, luz ambiente, informação para a criação da cena, etc. As propriedades do ambiente podem ser mostradas, ocultadas ou alteradas através do menu superior (File, Edit, Add...) na opção Tools e selecionando a seguir a opção Environment no V-REP. A tela de alteração da cena é mostrada na Figura 6.16, onde:

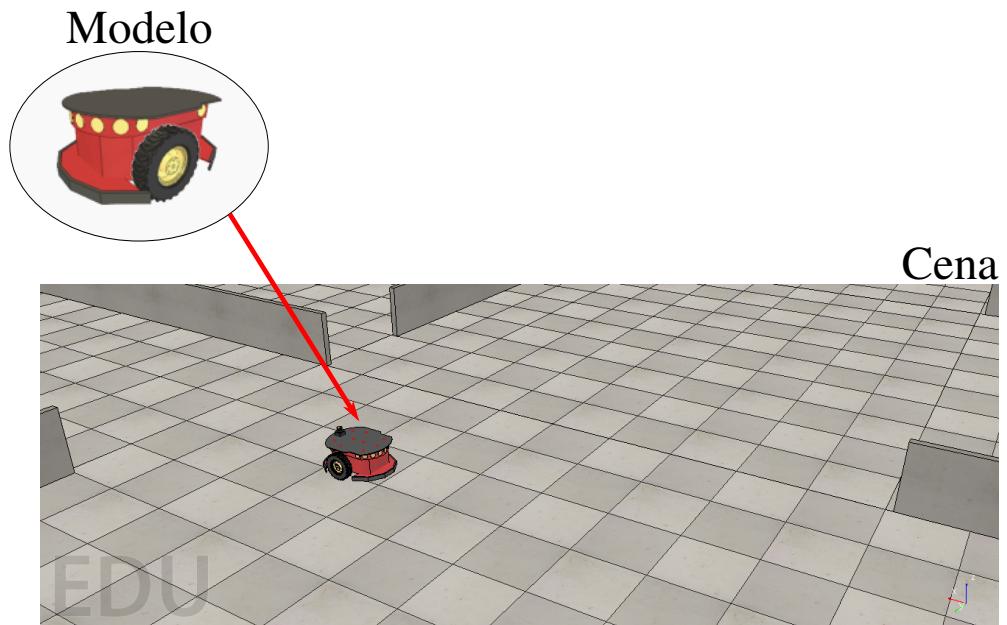


Figura 6.15. Modelo do robô Pioneer p3dx incluído na cena.

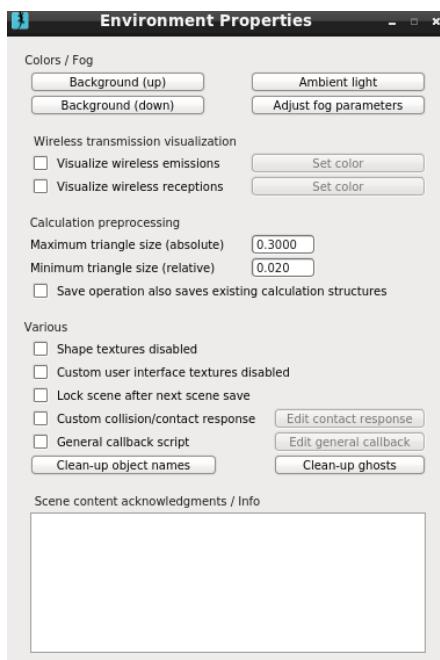


Figura 6.16. Propriedades do ambiente no V-REP.

- *Background (up / down)*: permite ajustar as cores de background da cena. Possui duas partes, a superior (céu) e os componentes correspondentes ao solo do ambiente.
- *Ambient light*: ajusta a luz da cena.

- *Adjust fog parameters*: permite adicionar nevoeiro no ambiente e assim pode-se dificultar a leitura e percepção do ambiente pelos sensores.
- *Visualize wireless emissions* e *Visualize wireless receptions*: habilita ou desabilita a visualização das comunicações sem fio.
- *Maximum triangle size* e *Minimum triangle size*: regem as dimensões aceitas para os triângulos empregados nos módulos de cálculos da simulação, mas que em nada altera a aparência dos objetos da cena. O tamanho máximo é absoluto, mas o tamanho mínimo do triângulo é relativo ao valor máximo aceito. Estes parâmetros afetam, por exemplo, o módulo de cálculo da distância mínima entre duas entidades, ou seja, quanto mais triângulos forem gerados para representar os objetos da cena maior será o custo computacional da cena.
- *Save operation also saves existing calculation structures*: opção que permite ao usuário do V-REP salvar, juntamente com a cena, as informações da estrutura de dados usada nos cálculos de colisão, distância, etc, assim é evitado algum tempo no pré-processamento da simulação. O efeito colateral é um arquivo de cena com mais informações e maior tamanho.
- *Shape textures disabled* e *Custom user interface textures disabled*: habilitam/desabilitam as texturas para formas e interfaces customizadas.
- *Lock scene after next scene save*: bloqueia a cena para edição, exportação de relatórios e visualização de *script*, sendo esse processo irreversível após o salvamento do arquivo.
- *Custom collision/contact response*: habilita/desabilita a customização do contato dinâmico da cena via *script*.
- *General callback script*: habilita/desabilita o retorno geral via *script*, ou seja, as chamadas feitas por um *plugin* da função da API *simHandleGeneralCallbackScript*.
- *Clean-up object names*: coloca um pouco de ordem nos nomes dos objetos.
- *Clean-up ghosts*: remove todos os objetos “fantasmas” que possam vir a estar ocultos na cena.
- *Scene content acknowledgements / Info*: permite colocar qualquer informação adicional textual que será exibida em toda abertura da cena.

6.4.2.2. Entidades

Entidade é o termo usado para um objeto ou uma coleção de objetos no V-REP.

Os objetos são os elementos usados para construir a cena da simulação, ou mesmo, na criação de modelos empregados nas cenas. Existem alguns tipos de objetos e eles podem ser facilmente reconhecidos pelo ícone presente ao lado de seu nome no componente que exibe a estrutura hierárquica dos objetos na cena (ver Componentes Hierárquicos na

Cena na Seção 6.4.1). A Figura 6.17 mostra os tipos de objetos presentes no V-REP e seus respectivos ícones.

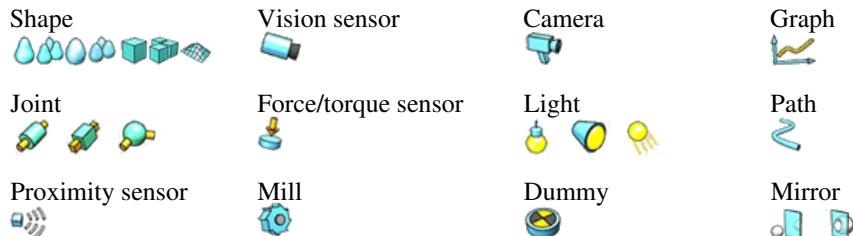


Figura 6.17. Tipos de objetos do V-REP [Coppelia Robotics 2015].

Os tipos de objetos presentes no V-REP são os seguintes:

- *Shape*: é uma forma rígida em malha composta por faces triangulares.
- *Joint*: é um atuador (ver Seção 6.2.2).
- *Graph*: é um gráfico de geração e visualização de dados da simulação.
- *Dummy*: é um ponto de orientação na cena.
- *Proximity sensor*: é um sensor que detecta a forma geométrica e volume de objetos da cena.
- *Vision sensor*: é uma câmera, ou seja, reage a luz, cores e imagens.
- *Force sensor*: sensor que mede forças e torques aplicados a ele.
- *Mill*: objeto capaz de efetuar operações de corte nos objetos shapes presentes na cena.
- *Camera*: objeto que permite a visualização da simulação por vários pontos de vistas.
- *Light*: ilumina a cena.
- *Path*: é um objeto do tipo caminho e que define uma trajetória.
- *Mirror*: é um objeto auxiliar que pode refletir imagens ou iluminação.

Alguns dos tipos de objetos possuem ainda propriedades especiais¹¹, as quais são levadas em conta pelos módulos de cálculos internos do V-REP, ou mesmo, regem algumas interações com outros objetos. As propriedades especiais são: *collidable* (objeto testado para colisão entre objetos), *measurable* (mensurável quanto a distância entre objetos), *detectable* (detectável por sensores de proximidade), *cuttable* (objeto cortável),

¹¹As propriedades especiais (*object special properties*) são acessíveis via menu do V-REP em *Tools / Scene object properties* e na janela aberta escolha a opção *Common*.

renderable (pode ser visto e detectado por sensores de visão) e *viewable* (compete a objetos que são transparentes ou que podem mostrar alguma imagem).

Uma coleção de objetos (*collections*) contém um grupo de objetos que são vistos pelo V-REP como uma única entidade. Ela serve de suporte para os módulos de cálculos onde só é levada em conta uma única entidade, portanto, possibilitando a um robô evitar o teste de colisão contra vários objetos (ao invés de um apenas). A coleção de objetos possui algumas das propriedades especiais disponíveis para objetos como: *collidable*, *measurable*, *detectable*, *cuttable* e *renderable*. Porém, as propriedades da coleção não são aplicadas a todos os objetos pertencentes a coleção, ou seja, caso uma coleção seja *collidable*, somente os objetos da coleção marcados também como *collidable* serão considerados para o cálculo da colisão.

6.4.3. Simulação

Alguns botões da Barra de Ferramentas 2 (ver Seção 6.4.1) auxiliam no controle da simulação, desempenhando papéis, como por exemplo, de iniciar, pausar ou parar a simulação. É possível ainda modificar algumas configurações na tela de configurações da simulação¹² no V-REP, que é mostrada na Figura 6.18, onde:

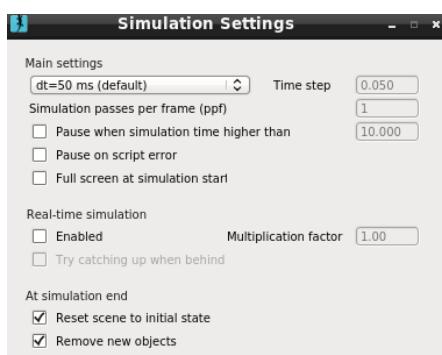


Figura 6.18. Configurações da simulação no V-REP.

- *Time step*: é o tempo decorrido da simulação em cada laço do *script* principal. Usando um tempo elevado a simulação torna-se mais rápida, porém, também apresenta-se mais instável e imprecisa. O inverso ocorre com tempos menores, a simulação torna-se mais precisa, porém mais lenta.
- *Simulation passes per frame (ppf)*: número de ciclos do *script* principal (simulação) para que seja refeita a tela da simulação, sendo comumente utilizado o valor 1 (a cada ciclo a tela é atualizada). Esta opção pode vir a ser interessante para placas gráficas com dificuldades de efetuar a visualização da simulação a contento.
- *Pause when simulation time higher than*: permite especificar em que momento a simulação será automaticamente pausada.

¹²A tela de configuração da simulação está acessível via menu do V-REP em Simulation / Simulation Settings.

- *Pause on script error*: quando habilitada a simulação é pausada automaticamente ao ocorrer erros no *script*.
- *Full screen at simulation start*: permite a execução da simulação em modo tela cheia. Durante a simulação tela cheia é possível voltar ao modo normal de exibição utilizando a tecla ESC do teclado ou via *script* alterando a variável booleana *sim_booparam_fullscreen* para falso.
- *Real-time simulation, multiplication factor*: se habilitada o V-REP tentará cumprir a execução da simulação em tempo real. E o *multiplication factor* dita quantas vezes mais rápido que o tempo real será o objetivo que o V-REP deverá perseguir na simulação.
- *Reset scene to initial state*: se selecionado, toda vez que a simulação é parada, todos os objetos voltam para o seu estado inicial, incluindo sua posição, orientação, posição dos atuadores, etc.
- *Remove new objects*: quando selecionado, qualquer objeto adicionado durante a simulação será automaticamente removido da cena ao final dela.

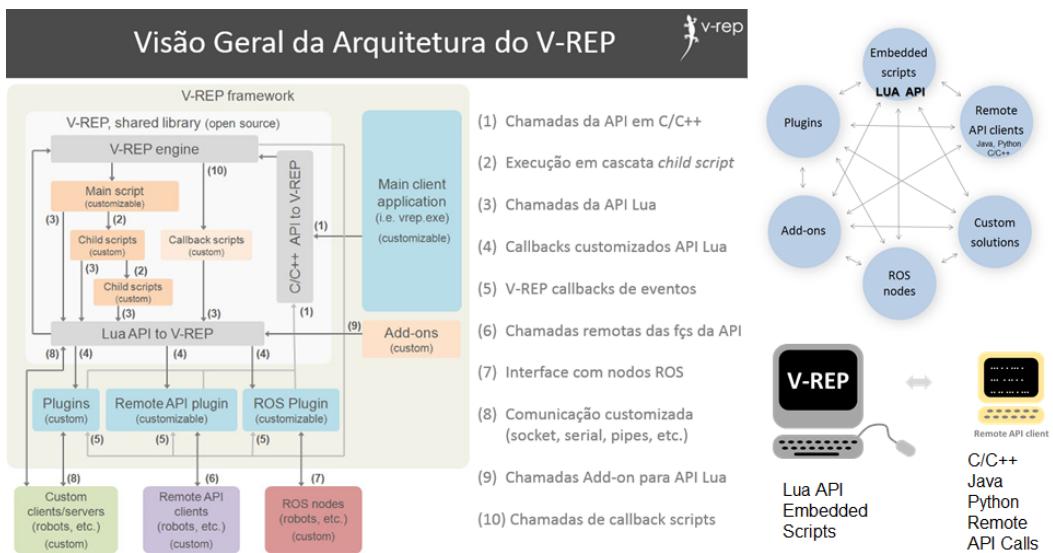


Figura 6.19. Arquitetura de Interfaces de Programação do V-REP. [Fonte: Adaptado de [E. Rohmer 2013] e V-REP Overview Presentation (Coppeliarobotics)]

6.4.4. V-REP e Programação

O V-REP é um simulador flexível e customizável, permitindo a programação de particularidades da simulação e o controle dos robôs (sensores e atuadores) utilizando, praticamente, qualquer linguagem de programação. A linguagem Lua [PUC-Rio 2015] (ver Seção 6.4.4.1) é a linguagem de programação nativa do V-REP, já estando incorporada ao simulador, permitindo a customização de toda a cena e manipulação de objetos através de seus scripts em LUA (*LUA API - Embedded Scripts*). Portanto, utilizando essa linguagem de programação se garante a compatibilidade de versões e a portabilidade de

sistema operacional da cena em V-REP, já que os *scripts* Lua são gravados juntamente com o salvamento da cena. Por outro lado, o V-REP oferece uma sofisticada arquitetura para o desenvolvimento de programas, através de diferentes interfaces, como apresentado na Figura 6.19. Portanto, existem outras formas de acesso à simulação via V-REP além dos scripts LUA, que são:

- *Add-on*: este método também utiliza Lua, porém utiliza um script em um arquivo externo à cena. Pode ser útil para casos em que se queira customizar rapidamente o simulador, pois, trocando o arquivo *script* Lua já se pode mudar o comportamento de um robô, por exemplo. O *script* Lua nesse método é iniciado automaticamente com a simulação.
- *Plugin*: pode-se desenvolver plugins para V-REP utilizando as linguagens C/C++. Um plugin é uma biblioteca que é automaticamente incorporada ao V-REP. Um *plugin* pode funcionar como extensão da linguagem lua, permitindo assim a criação de novos comandos e facilitadores para seus *scripts*. Em outros casos, pode ser usado para ganhar desempenho já que este método é o mais eficiente em termos computacionais. Pode-se desenvolver ainda um *plugin* para controlar toda a cena, como é feito com o Lua.
- *API remota*: um cliente remoto da API pode ser desenvolvido em C/C++, em Python, Java, Matlab/Octave, em um script Urbi, ou mesmo através de um *script* Lua. A API remota disponibiliza cerca de 100 funções que permitem o controle da cena e dos objetos presentes nela, utilizando para isso comunicação TCP/IP e chamadas de funções equivalentes/similares a da API Lua embutida no V-REP. O programa controlador da cena não será automaticamente inicializado, como nos casos anteriores, ou seja, é necessária a inicialização da cena no V-REP e então a execução do programa controlador da cena. Esta opção permite que o programa controlador esteja locado em qualquer computador com acesso a internet.
- *ROS node*: este método permite que uma aplicação externa conecte o V-REP usando o ROS (*Robot Operating System*), suportando assim, qualquer linguagem de programação que utilizar o ROS. A aplicação ROS não precisa necessariamente estar na mesma máquina que a simulação.
- *Cliente/servidor customizado*: pode-se criar ainda um método próprio de interação com o V-REP, mas neste caso o cliente ou o servidor precisam se comunicar com o V-REP utilizando um *script*, *plugin* ou utilizando algum meio de comunicação (por exemplo, *sockets*). Este é o meio mais flexível por permitir ao V-REP interagir com qualquer linguagem de programação. Porém é o método mais trabalhoso já que toda a estrutura de comunicação precisa ser recriada ou adaptada. Esta abordagem permite a criação de uma arquitetura cliente-servidor entre o V-REP e um módulo externo de controle dos robôs.

Mesmo utilizando uma linguagem diferente do Lua para a customização do V-REP, é conveniente ao usar o V-REP ter noções básicas dessa linguagem de programação,

uma vez que o LUA é a linguagem nativa de desenvolvimento junto ao V-REP. Um exemplo dessa utilidade vem da necessidade de se editar os scripts padrão de certas cenas visando eliminar comportamentos padrões pré-existentes em alguns modelos do V-REP. Portanto, caso se deseje controlar um objeto proveniente deste modelo fora do V-REP, será necessário antes apagar partes do código Lua padrão.

Nas subseções a seguir são mostradas algumas informações adicionais da linguagem Lua e de seu uso junto ao V-REP, e em seguida são apresentadas informações adicionais sobre a programação do V-REP utilizando a API remota.

6.4.4.1. Lua

Lua [PUC-Rio 2015] é uma linguagem *script* e procedural, que foi criada com o intuito de ser um facilitador para expansão de aplicações. Lua é uma linguagem case sensitive, ou seja, faz diferença entre letras maiúsculas e minúsculas. Sua criação se deu por uma equipe de desenvolvedores da Tecgraf da PUC-Rio.

As variáveis em Lua são dinamicamente tipadas, ou seja, o dado contido nela determina seu tipo. Os tipos básico de dados que uma variável pode conter são: *nil* (valor nulo e é atribuído automaticamente na criação da variável), *boolean* (*true/false*), *number* (números reais), *string* (conjunto de caracteres de 8 bits), *function* (apontamento para funções em Lua), *userdata* (bloco de memória que armazena qualquer tipo de dado), *thread* (variável que representa fluxos de execução independentes de rotinas Lua) e *table* (são arrays que podem conter dados de todos os tipos exceto *nil*). As variáveis podem ser globais, locais ou campos de tabelas, sendo que o Lua utiliza por padrão uma variável como global. Em Lua pode-se efetuar múltiplas atribuições de variáveis, um exemplo dessa utilização é mostrada na Figura 6.20.

```
x, y, z = myTable[1], myTable[2], myTable[3]
```

Figura 6.20. Múltiplas atribuições em Lua.

Os operadores relacionais em Lua são: `==` (igualdade), `~=` (negação de igualdade), `<` (menor que), `>` (maior que), `<=` (menor ou igual que) e `>=` (maior ou igual que). Na Figura 6.21 é mostrado um exemplo de controle condicional (*if*) em Lua usando o operador de igualdade. Pode-se encadear cláusulas usando as palavras-chaves *and* (E) ou *or* (OU). Usando `--` pode-se declarar todo o texto até o final dessa linha como comentário, ou mesmo, empregar `--[[` para abrir e `--]]` para fechar o texto de comentário a ser ignorado pelo interpretador Lua. Os laços que podem ser utilizados em Lua são *for*, *while* e *repeat*. Na Figura 6.22 são mostrados exemplos dos laços onde são exibidos no terminal os números de 1 a 4. Nota: no V-REP devemos considerar em qual console/janela serão exibidos os dados, e muitas vezes são usadas funções da API do V-REP para esta finalidade.

Um exemplo de *script* em Lua do V-REP é mostrado na Figura 6.23, onde, é efetuado o controle de um robô Pioneer baseado no sensor Sonar/Ultrassom (*usensor*) para gerar comandos de desvio de obstáculos usando cinemática diferencial (controle indepen-

```

if value1==value2 then
    print('value1 and value2 are the same!')
end

```

Figura 6.21. Controle condicional em Lua

```

--Contando de 1 a 4 usando for.
for i=1,4,1 do
    print(i)
end

--Contando de 1 a 4 usando while.
i=0
while i~=4 do
    i=i+1
    print(i)
end

--Contando de 1 a 4 usando repeat.
i=0
repeat
    i=i+1
    print(i)
until i==4

```

Figura 6.22. Opções de Laço em Lua

dente de velocidade dos motores *vLeft* e *vRight*). Este é o código do script Lua padrão associado ao robô Pioneer P3DX disponível junto aos exemplos de modelos de robôs do V-REP. O script apresentado na Figura 6.23, como é usual nos scripts dos objetos simulados, é composto por 3 partes: Inicialização (*sim_childscriptcall_initialization*), Finalização (*sim_childscriptcall_cleanup*) e Execução (*sim_childscriptcall_actuation* no caso dos atuadores, podendo ter também um *sim_childscriptcall_sensing* no caso dos sensores). Também é importante destacar que alguns scripts possuem parâmetros que definem as propriedades dos elementos, como por exemplo, nos sensores laser Sick ou Hokuyo onde a abertura de varredura é definida através destes parâmetros (ajustados clicando no ícone ao lado direito do ícone do script Lua do objeto).

Apesar do Lua ser uma linguagem procedural, o V-REP faz chamadas nos scripts de inicialização, finalização e execução, utilizando tipos de chamadas (*sim_call_type*), ou seja, o V-REP utiliza uma programação por eventos, onde, controles condicionais encapsulam os comandos específicos dos eventos da simulação. No caso de uma simulação que inclua diferentes objetos, como por exemplo, um robô Pioneer P3DX dotado com um sensor laser Hokuyo URG-04 e um GPS, cada um destes elementos terá um script Lua próprio associado a ele.

```

1 -- This is a very simple EXAMPLE navigation program,
2 -- which avoids obstacles using the Braitenberg algorithm
3
4 if (sim_call_type==sim_childscriptcall_initialization) then
5
6 if (sim_call_type==sim_childscriptcall_cleanup) then
7
8 end
9
10 if (sim_call_type==sim_childscriptcall_actuation) then
11   for i=1,16,1 do
12     res,dist=simReadProximitySensor(usensors[i])
13     if (res>0) and (dist<noDetectionDist) then
14       if (dist<maxDetectionDist) then
15         dist=maxDetectionDist
16       end
17       detect[i]=1-((dist-maxDetectionDist)/(noDetectionDist-maxDetectionDist))
18     else
19       detect[i]=0
20     end
21   end
22
23   vLeft=v0
24   vRight=v0
25
26   for i=1,16,1 do
27     vLeft=vLeft+braitenbergL[i]*detect[i]
28     vRight=vRight+braitenbergR[i]*detect[i]
29   end
30
31   simSetJointTargetVelocity(motorLeft,vLeft)
32   simSetJointTargetVelocity(motorRight,vRight)
33
34 end
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

Figura 6.23. Exemplo de *script* em Lua para o controle de um robô Pioneer usando o Sonar/Ultrassom (*usensor*) para gerar comandos de desvio de obstáculos em um robô com cinemática diferencial (controle independente de velocidade dos motores *vLeft* e *vRight*).

Um elemento importante para passar valores (variáveis) de um script para outro em Lua dentro de uma simulação do V-REP é o uso dos *tubes* ou de *signals*. Por exemplo, se desejamos ler os dados de um sensor laser Sick e enviar estes dados para outro script responsável pela ativação dos motores do robô, podemos passar estes dados através de um *communicationTube* ou *Set/Get-Signal*. A Figura 6.24 apresenta um exemplo do uso de signals para comunicar o script do laser com o script do robô.

Um outro elemento importante na interação com o V-REP é a leitura de eventos do teclado, permitindo assim controlar robôs pelo teclado ou ativar algum tipo de funcionalidade durante a execução da simulação. A Figura 6.25 apresenta um exemplo de leitura do teclado, com a exibição das teclas lidas na barra de status. A geração de arquivos de *log* também é uma outra funcionalidade importante para o desenvolvimento de aplicações robóticas, podendo ser implementado de modo direto através de comandos da própria linguagem Lua.

6.4.4.2. API remota

O V-REP permite diferentes modos de programação e comunicação com o simulador. A API remota é um destes modos, e trabalha com o conceito cliente/servidor, do lado do servidor disponibilizando as informações da cena e para o cliente possibilitando seu controle através de chamadas a API. Um arquivo de cena com exemplos dos diferentes modos de programação e comunicação com o V-REP é disponibilizado na pasta “scenes”,

```

Non-threaded child script (LaserScanner_2D)
45 mindst=10.0
46 for i=0,pnts,1 do
47     simSetJointPosition(jointHandle,p)
48     p=p+stepSize
49     r,dist,pt=simHandleProximitySensor(laserHandle) -- pt is RELATIVE to te rotating laser beam!
50     if r>0 then
51         if (dist < mindst) then
52             mindst=dist
53         end
54     end
55     -- We put the RELATIVE coordinate of that point into the table that we will return:
56     m=simGetObjectMatrix(laserHandle,-1)
57     pt=simMultiplyVector(m,pt)
58     pt=simMultiplyVector(modelInverseMatrix,pt) -- after this instruction, pt will be relative
59     end
60     simHandleGraph(graphHandle,0.0)
61 end
62 simSetFloatSignal("LaserFrontal",mindst) -- Signal: minimum distance
63

Non-threaded child script (Pioneer_p3dx)
21 if (sim_call_type==sim_childscriptcall_actuation) then
22     vLeft=2.0 -- Move Forward
23     vRight=2.0
24
25     -- READ THE DATA:
26     valordist=simGetFloatSignal("LaserFrontal") -- Read Signal
27     if (valordist) then
28         if (valordist < 0.5) then
29             vLeft=-2.0 -- Turn Left
30             vRight=2.0
31         end
32     end
33     simSetJointTargetVelocity(motorLeft,vLeft)
34     simSetJointTargetVelocity(motorRight,vRight)
35 end
36

```

Figura 6.24. Exemplo de *scripts* em Lua com um *signal* para comunicar dados entre os *scripts*.

```

Non-threaded child script (Manta)
49 -- Read the keyboard messages (make sure the focus is on the main window, scene view):
50 message,auxiliaryData=simGetSimulatorMessage()
51
52 while message~= -1 do
53     if (message==sim_message_keypress) then
54         simAddStatusbarMessage(string.format("Tecla: %d",auxiliaryData[1]))
55
56         if (auxiliaryData[1]==2007) then -- up key
57             if (motor_velocity<dVel*9.99) then
58                 motor_velocity=motor_velocity+dVel
59             end
60         end
61         if (auxiliaryData[1]==2008) then -- down key
62             if (motor_velocity>-dVel*9.99) then
63                 motor_velocity=motor_velocity-dVel
64             end
65         end
66         if (auxiliaryData[1]==2009) then -- left key
67             if (motor_velocity>0) then
68                 motor_velocity=motor_velocity-dVel
69             end
70         end
71         if (auxiliaryData[1]==2010) then -- right key
72             if (motor_velocity<0) then
73                 motor_velocity=motor_velocity+dVel
74             end
75         end
76     end
77     message,auxiliaryData=simGetSimulatorMessage()
78 end
79
80

```

Figura 6.25. Exemplo de *scripts* em Lua com a leitura de teclas.

denominado de “controlTypeExamples.ttt”, o qual inclui um exemplo de robô controlado através da API remota.

A primeira etapa para a sua utilização é a habilitação do servidor na simulação. Essa habilitação se dá através do comando *simExtRemoteApiStart(#Porta)* inserido no *script* principal Lua (ver Seção 6.4.4.1) da simulação, na condição de inicialização. A Figura 6.26 mostra um exemplo de como inicializar o servidor da API na porta 19999.

No lado do cliente é preciso acessar as funções da API remota para possibilitar a interação com a simulação. Em C/C++, as funções da API são incluídas com a adição dos

```

if (sim_call_type==sim_mainScriptCall_initialization) then
    simOpenModule(sim_handle_all)
    simHandleGraph(sim_handle_all_except_explicit,0)

    -- habilitando servidor na porta 19999
    simExtRemoteApiStart(19999)
end

```

Figura 6.26. Habilitação do servidor da API remota na simulação do V-REP na porta 19999.

arquivos¹³ “extApi.h”, “extApi.c”, “extApiPlatform.h” e “extApiPlatform.c” ao projeto da aplicação.

Para outras linguagens, é disponibilizada uma biblioteca¹⁴ constituída com base na API de C/C++, possibilitando acesso à simulação, em qualquer linguagem que possa incorporar esta biblioteca.

Cerca de 100 funções estão disponíveis¹⁵ para acesso a simulação. Todas as funções são facilmente reconhecidas pelo prefixo “simx”. A função que faz o estabelecimento da conexão com o servidor e, portanto, será sempre a primeira a ser utilizada é a *simxStart*, onde são passados os parâmetros de comunicação, como por exemplo, IP e porta do servidor.

6.4.4.3. V-REP, Matlab e o Robotics Toolbox

O simulador V-REP disponibiliza uma API remota para Matlab, permitindo o desenvolvimento de aplicações cliente para controle de robôs móveis e manipuladores.

Como requisito para utilização do Matlab com V-REP, é necessário que se copie para o diretório em que será executado o programa cliente desenvolvido em Matlab, a biblioteca da API remota do V-REP apropriada para o sistema operacional cliente (“remoteApi.so” para linux, “remoteApi.dll” para windows, ou “remoteApi.dylib” para Mac)¹⁶. Se for possível escolher entre uma versão 32-bit ou 64-bit deste arquivo, deve-se escolher a versão que corresponde a instalação do Matlab (32-bit ou 64-bit). Além da biblioteca da API remota, também devem ser copiados os arquivos “remAPI.m” e “remoteApiProto.m” disponibilizados pelo V-REP¹⁷. Um exemplo de como criar uma função em Matlab para acesso a simulação do V-REP também é disponibilizado junto aos arquivos citados acima (“simpleTest.m”).

¹³Disponibilizados junto com a instalação do V-REP e acessíveis pelo caminho programming/remoteApi/ (partindo-se da pasta raiz do V-REP).

¹⁴Deve-se usar a biblioteca da API remota apropriada para o sistema operacional cliente, sendo, “remoteApi.dll” para Windows, “remoteApi.dylib” para Mac e “remoteApi.so” para Linux. Partindo-se da pasta de instalação do V-REP é possível obter a biblioteca em programming/remoteApiBindings/lib/lib/.

¹⁵A documentação da API remota pode ser obtida no manual do V-REP que está disponível em <http://www.coppeliarobotics.com/helpFiles/>.

¹⁶Procurar na pasta programming/remoteApiBindings/lib/lib

¹⁷Procurar na pasta programming/remoteApiBindings/matlab/matlab

A figura 6.27 mostra um exemplo de código em Matlab para iniciar a comunicação entre o cliente e o V-REP, e iniciar a simulação. Antes de iniciar a comunicação com o Matlab é preciso que uma simulação (cena) seja carregada no V-REP e o servidor da API Remota esteja habilitado nesta simulação. O arquivo de cena “controlTypeExamples.ttt”, localizado na pasta de “scenes” do V-REP inclui um exemplo de como ativar o servidor da API remota e controlar um robô através desta API remota.

```
vrep=remApi('remoteApi');
id = vrep.simxStart('127.0.0.1', 19999, true, true, 2000, 5);
res = vrep.simxStartSimulation(id, vrep.simx_opmode_oneshot_wait);
```

Figura 6.27. Exemplo de como iniciar a conexão do cliente em Matlab com o simulador V-REP executado na máquina local, IP 127.0.0.1, na porta 19999.

As mesmas funções da API que estão disponíveis para outras linguagens (C/C++, Python, Java e LUA), também estão disponíveis para uso no Matlab, e também possuem o prefixo “simx”. Por exemplo, “simxSetJointTargetPosition” e “simxSetJointTargetVelocity” podem ser usadas para definir a posição e velocidade desejada para as juntas de um robô manipulador, e “simxGetObjectPosition” e “simxGetObjectOrientation” para obter a posição e orientação de um objeto, incluindo o efetuador de um robô manipulador.

A *Robotics Toolbox* para Matlab [Corke 2011] já possui um conjunto de funções e algoritmos implementados que auxiliam no estudo e desenvolvimento de sistemas de controle para robôs móveis e manipuladores. Esta toolbox fornece, por exemplo, funções para modelar e simular a cinemática e dinâmica de robôs manipuladores e veículos não-holonômicos, gerar trajetórias e planejar rotas, lidar com transformações homogêneas de sistemas de coordenadas, algoritmos para localização, mapeamento, SLAM (*Simultaneous Localization and Mapping*), dentre outras. O livro [Corke 2011] apresenta diversos exemplos destas funções em Matlab usadas para o desenvolvimento de aplicações e pesquisas em robótica.

Devido a variedade de funções implementadas na *Robotics Toolbox* para Matlab, o uso combinado desta *toolbox* com o simulador V-REP se mostra como uma solução atrativa para a prototipagem rápida de um sistema de controle e percepção para robôs. Naturalmente, estas ferramentas também podem ser utilizadas em cursos de robótica a nível de graduação e pós-graduação. Tento em vista essa motivação no ensino da robótica, Renaud Detry criou o projeto *TRS: An Open-source Recipe for Teaching/Learning Robotics with a Simulator*, disponível em <http://ulgrobotics.github.io/trs> para um curso na Universidade de Liège. Este projeto de código aberto fornece um ambiente de simulação para V-REP (arquivo “.ttt”), um conjunto básico de funções para Matlab, exemplos de código, e uma estrutura de projeto, com objetivo, documentação e metas intermediárias, para ser executado por alunos de um curso de robótica de nível de mestrado. O TRS foi apresentado em um *workshop* do ICRA¹⁸ em 2014, e um tutorial do TRS foi realizado no IROS¹⁹ em 2014, com previsão de tutoriais a serem apresentados no ICVS²⁰

¹⁸IEEE International Conference on Robotics and Automation

¹⁹IEEE/RSJ International Conference on Intelligent Robots and Systems

²⁰International Conference on Computer Vision Systems

2014, Ro-Man²¹ 2015 e IROS 2015. Portanto, é possível notar o crescente interesse da comunidade de robótica para com o simulador V-REP, e em particular, a grande quantidade de possibilidades que são abertas pelo fato de poder conectar o V-REP com este *toolbox* do Matlab.

6.4.4.4. V-REP e ROS (*RObot Operating System*)

O V-REP não poderia deixar de oferecer uma interface para com a plataforma ROS (*RObot Operating System*²²), uma das ferramentas mais conhecidas e utilizadas por quem desenvolve aplicações com robôs, principalmente com robôs reais. O ROS é um *framework* para o desenvolvimento de aplicações robóticas baseado em uma arquitetura de serviços robóticos, e que muitas vezes é visto como um “sistema operacional robótico”, uma vez que provê um *middleware* de acesso a diferentes plataformas robóticas, assim como diversas aplicações de alto-nível (pacotes ROS) que implementam comportamentos e funcionalidades (p.ex. planejamento, auto-localização, navegação) para robôs. O ROS pode ser usado junto com simuladores, como por exemplo o Gazebo (ver 6.3.1 e 6.3.2) e também com o V-REP.

O ROS permite o controle de robôs reais ou simulados via *ROS nodes*. O V-REP oferece uma interface de comunicação com o ROS, que é naturalmente um sistema distribuído orientado a serviços, e assim, é possível comunicar o V-REP com o ROS, através de nodos que realizam trocas de mensagens. O V-REP provê a publicação de mensagens via *V-REP ROS publishers* e a assinatura de mensagens via *V-REP ROS subscribers*, que são mecanismos familiares a quem trabalha com os serviços de publicação e assinatura de mensagens dos nodos do ROS. Entretanto, o próprio ROS e a comunicação com o ROS são mecanismos mais complexos e que requerem um estudo mais aprofundado da interface ROS para o desenvolvimento de aplicações que façam uso desta abordagem.

O arquivo exemplo de cena “controlTypeExamples.ttt” também inclui um exemplo de controle de um robô através do ROS, onde é necessário ter o ROS instalado, configurado e preparado para se comunicar com o robô deste exemplo.

6.5. Aplicações da Robótica Móvel e Inteligente

Nesta seção, são apresentados exemplos de aplicações de robótica móvel e inteligente, relacionando o seu projeto e desenvolvimento, com a possibilidade de realizar trabalhos através da implementação de simulações.

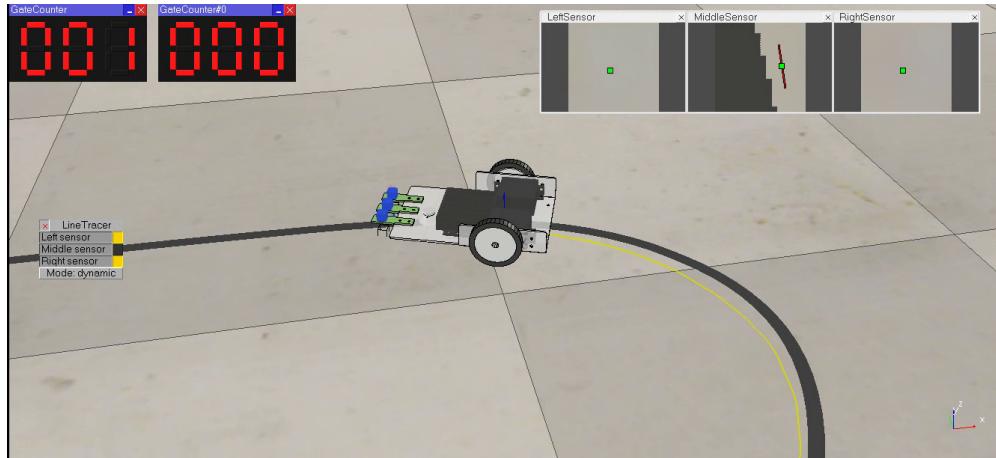
Uma primeira aplicação robótica muito presente em competições de robótica (como a OBR²³, por exemplo), por ser relativamente simples de ser implementada e, também, não muito dispendiosa em termos de sensores, são os robôs seguidores de linhas. As linhas são demarcadas, habitualmente feitas no piso, e servem como rotas pré-planejadas de deslocamento. Os robôs tem um comportamento autônomo (sem intervenção humana), seguindo a linha que define sua trajetória.

²¹International Symposium on Robot and Human Interactive Communication

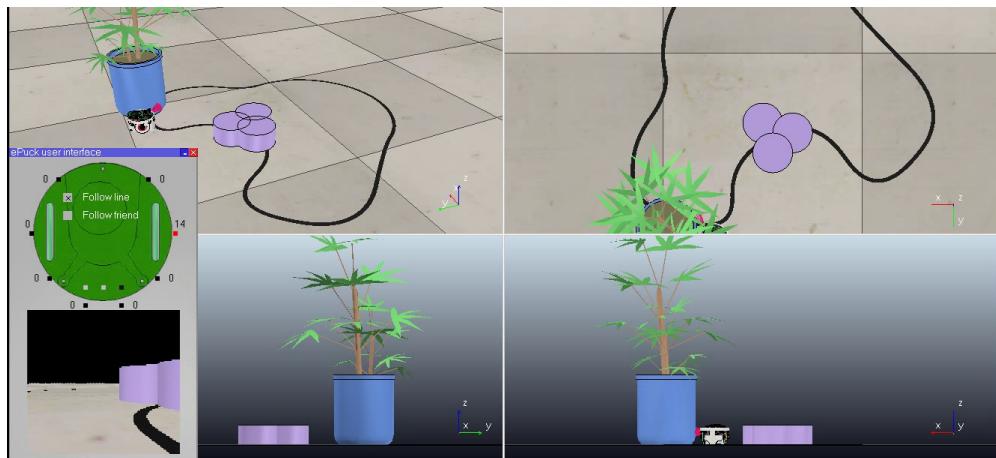
²²ROS: <http://www.ros.org/>

²³OBR - Olimpíada Brasileira de Robótica: <http://www.obr.org.br/>

Um exemplo de seguidor de linha²⁴ utilizando V-REP é mostrado na Figura 6.28(a), onde, um robô é equipado com três sensores (câmeras) focados no solo, visando a detecção das linhas. Quando a linha é detectada pelo sensor da esquerda, o robô é rotacionado para a esquerda e o inverso ocorrendo ao ser detectada a linha pelo sensor da direita. O objetivo do robô seguidor de linha é manter a linha (região escura) com a maior área possível junto ao sensor central, onde nessa situação o robô deve seguir em frente.



(a) Seguidor de linha 1 (cena “LineTracer-threaded.ttt”)



(b) Seguidor de Linha 2 (cena “e-puckDemo.ttt”)

Figura 6.28. Exemplos de robôs preparados para seguir linhas.

A Figura 6.28(b) mostra um segundo exemplo de robô seguidor de linha²⁵, um pouco mais complexo. Este robô é capaz de contornar obstáculos (usando um sensor de proximidade) que estejam sobre as linhas, tornando-se, portanto, mais robusto para uma aplicação real. Neste exemplo ainda, mostra-se a cena de quatro pontos de vistas diferentes.

²⁴O vídeo da simulação no V-REP do seguidor de linhas 1 está disponível em https://youtu.be/ij_UVMWpQIs.

²⁵O vídeo do seguidor de linhas 2 está disponível no endereço <https://youtu.be/wf9n1PrRvW4>.

Um robô seguidor de linha é, portanto, um robô estritamente reativo que é capaz de caminhar por rotas previamente definidas. Robôs seguidores de linhas são comuns em competições robóticas, mas também existem produtos comerciais baseados neste tipo de robôs, como o OZOBOT [Evollve 2015]. Inclusive alguns robôs móveis industriais, como os AGVs (*Automated Guided Vehicles*) da Kiva Systems [Kiva Systems 2015], são seguidores de linhas que foram adotados pela Amazon para a distribuição de mercadorias em seus armazéns.

Outro comportamento reativo similar ao seguidor de linhas é o seguidor de paredes. Com essa estratégia usualmente é possível percorrer todos os cômodos de um ambiente fechado, por isso pode ser empregado para robôs que efetuam o mapeamento de ambientes. Um exemplo de comportamento de seguidor de paredes²⁶ é mostrado na Figura 6.29. Neste exemplo, um robô Pioneer P3dx [Adept MobileRobots 2015] com GPS, um sensor *laser* Sick [SICK 2015] e três câmeras desempenha comportamentos de seguidor de paredes (Sick) e Seguidor de linhas (câmeras e Sick para desvio de obstáculos). O robô inicia em um ambiente interno onde segue as paredes utilizando seu sensor Sick para manter uma distância mínima e máxima da parede (objeto) e verificando paredes e objetos a frente. Este comportamento permanece até encontrar a porta de saída para o ambiente externo, passando assim, a ser um robô seguidor de linhas. O objetivo dessa simulação é fazer com que o robô alcance uma coordenada específica do ambiente (onde uma pessoa está sentada aguardando sua chegada) usando somente a combinação de comportamentos reativos: seguir paredes, desviar de obstáculos, seguir linhas, seguir em direção a uma coordenada alvo.

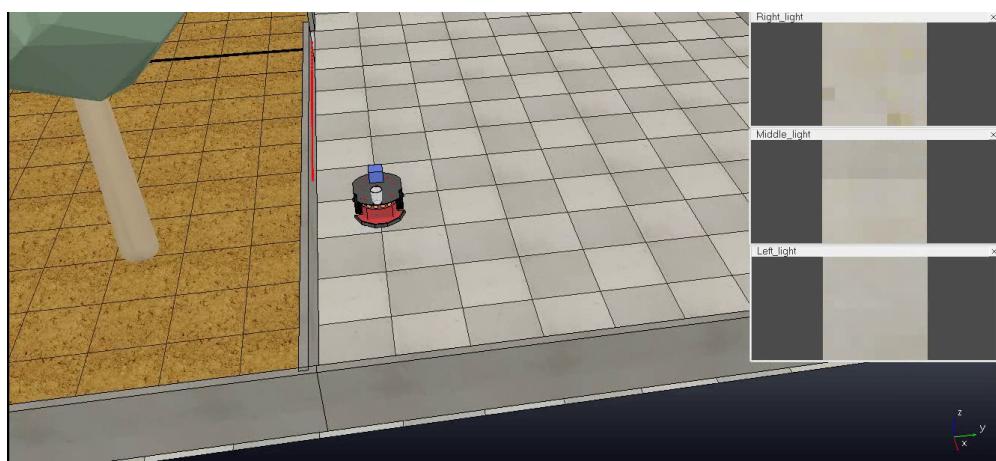


Figura 6.29. Robô que executa comportamentos reativos: seguir paredes e linhas.

Pode ser interessante para um robô desempenhar um comportamento de perseguição, ou seja, seguir algum outro robô ou objeto em deslocamento. Esse comportamento é reativo também. Na Figura 6.30, mostra-se um exemplo de simulação em que um robô hexapod²⁷ (robô de seis “pernas”) branco persegue um outro hexapod vermelho utilizando

²⁶O vídeo do seguidor de paredes e linhas está disponível no endereço <https://youtu.be/-1KCFHhnSjA>.

²⁷O vídeo da simulação de robô perseguidor no V-REP está disponível em <https://youtu.be/zn9KSZZNenI>.

uma câmera. O perseguidor procura no ambiente pela cor vermelha em uma tonalidade e brilho específicos e busca manter a região encontrada dentro de certas dimensões e centralizada. Portanto, caminha à frente sempre que nota que a largura da região segmentada do robô perseguido é menor que certo valor; caminha para trás quando percebe que é maior que o valor máximo aceito; e busca também deixar o centro de massa da região segmentada próximo ao centro horizontal da imagem fornecida pela câmera.

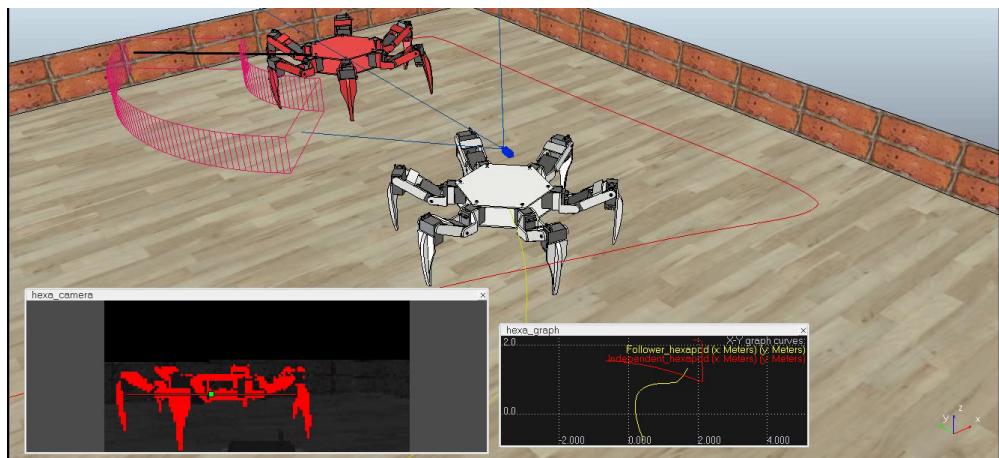


Figura 6.30. Robô que efetua o comportamento reativo de perseguidor utilizando a cor, sendo, o robô vermelho perseguido pelo branco (cena “imageProcessingExample.ttt”).

É possível também se obter o comportamento adequado de um robô utilizando um algoritmo evolutivo de seleção natural. Na Figura 6.31, é mostrado um exemplo desse tipo de abordagem²⁸, onde 5 indivíduos são mantidos em simulação a cada instante. Quando o V-REP detecta que o robô colidiu ele é eliminado e um outro indivíduo é gerado com base nos melhores da população, sendo os melhores aqueles que mais se locomovem pelo ambiente sem colisões. O mais interessante é que o robô não é programado manualmente para reagir às informações obtidas pelos sensores, e sim, à medida que as gerações passam, um comportamento interessante emerge sendo obtido pela evolução dos robôs simulados. A obtenção de comportamentos por evolução e seleção natural só se torna viável em ambientes simulados fiéis (com um motor físico consistente, robôs e sensores fiéis à realidade), e com um grande número de robôs e/ou repetições. Só assim, após um longo processo de “evolução” (simulada) este comportamento poderá ser confiável suficiente para ser utilizado em um ambiente real.

O mapeamento de ambientes é um outro problema importante na robótica. Este mapeamento permite que um sistema robótico crie um mapa e depois planeje suas ações com base em uma descrição fidedigna do ambiente gerada neste processo. Com o V-REP pode-se desenvolver e aprimorar algoritmos de mapeamento²⁹. A Figura 6.32 mostra a criação de uma nuvem de pontos descrevendo o ambiente (mapa 3D) utilizando um sensor

²⁸O vídeo que mostra a utilização da seleção natural está disponível em <https://youtu.be/LTOnQYS60Fo>.

²⁹O vídeo de um exemplo de mapeamento 3D de um ambiente e está disponível no endereço <https://youtu.be/IP14qvYzqtw>.

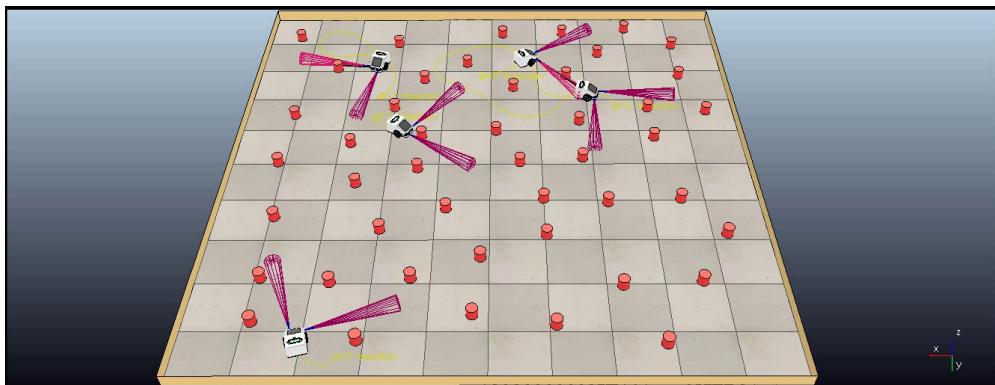


Figura 6.31. Utilizando o V-REP e algoritmo de seleção natural para encontrar comportamento adequado para robôs (cena “naturalSelectionAlgo.ttt”).

Hokuyo [Hokuyo Automatic 2015] sendo rotacionado e inclinado por uma base (*pan/tilt*), que, por sua vez, está posicionada sobre um robô móvel com esteiras. A nuvem de pontos gerada é exibida em azul na janela superior direita da tela, enquanto a propagação do *laser* do Hokuyo é mostrada em vermelho na cena. O sensor Hokuyo é 2D (mapeia um corte planar da cena), por isso torna-se necessária sua inclinação e giro para possibilitar a captura do mapa 3D da cena. A grande vantagem de se usar um sensor como o Hokuyo nesta tarefa é seu preço, ou seja, caso o algoritmo seja utilizado em um ambiente real, o custo de constituição desse sistema seria menor do que a aquisição de um sensor *laser* 3D, como por exemplo, o Velodyne [Velodyne LiDAR 2015].

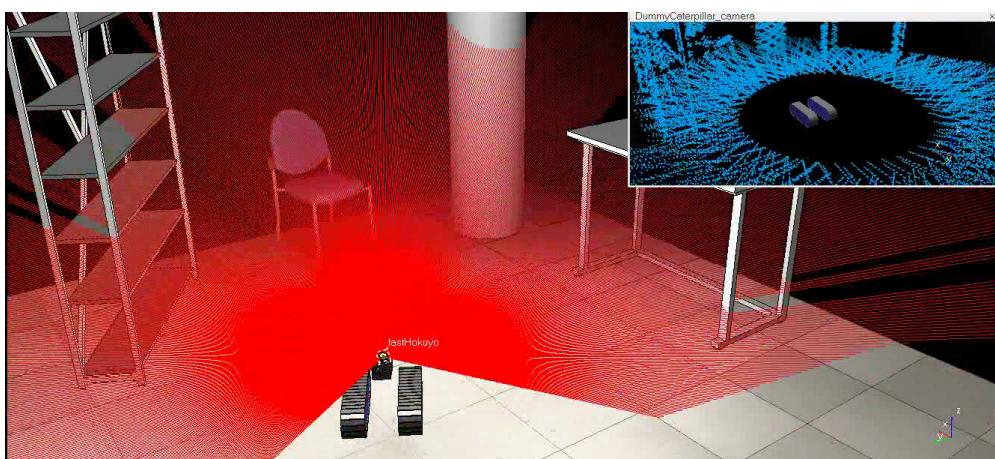
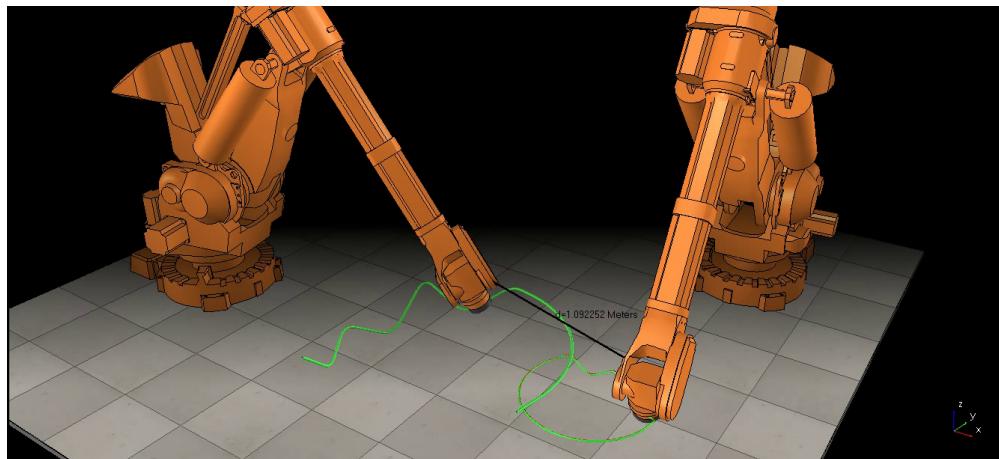


Figura 6.32. Mapeamento 3D do ambiente utilizando um Hokuyo e um suporte Pan/Tilt (cena “environmentMapping.ttt”).

O V-REP também pode ser útil para aplicações industriais que utilizem braços robóticos. Pode-se testar rotas, prevenindo-se de eventuais colisões, antes da utilização na prática em um ambiente real. O V-REP possui algumas rotinas próprias de detecção de colisões, distância entre objetos, formando um arcabouço matemático eficaz e de fácil implementação. A Figura 6.33(a) apresenta um teste efetuado com dois braços robóticos³⁰

³⁰A execução do teste é mostrado no endereço <https://youtu.be/Q8kgWcxYaY8>.

que percorrem suas rotas (em verde) e a distância mínima entre eles é monitorada. Já a Figura 6.33(b) apresenta dois braços robóticos³¹ efetuando movimentos planejados e sincronizados no desempenho da tarefas de mover bolinhas de um copo a outro, ou mesmo, empilhar copos.



(a) Planejamento e teste de Rotas (cena “2IndustrialRobots.ttt”)



(b) Planejamento de movimentos (cena “motionPlanningAndGraspingDemo.ttt”)

Figura 6.33. V-REP no planejamento e teste de rotas/movimentos com braços robóticos.

Outra funcionalidade interessante que o V-REP possibilita é a retirado (corte) de material, em que os cortes se dão apenas em alguns objetos especiais declarados com a propriedade cortável (ver Seção 6.4.2.2). É possível, portanto, colocar uma fresa acoplada a um braço robótico³² e simular a sua atuação no corte do material, buscando assim sanar eventuais problemas na programação dos movimentos do braço robótico e, ainda, visualizando o resultado final do corte. Na Figura 6.34, a simulação da usinagem de uma peça de aço é mostrada.

³¹O vídeo que mostra a exibição da simulação dos movimentos dos braços robóticos está disponível em <https://youtu.be/B16v-uRnbuc>.

³²Vídeo exemplo de braço robótico com fresa no V-REP está disponível em <https://youtu.be/4aEhAd5EhbQ>.

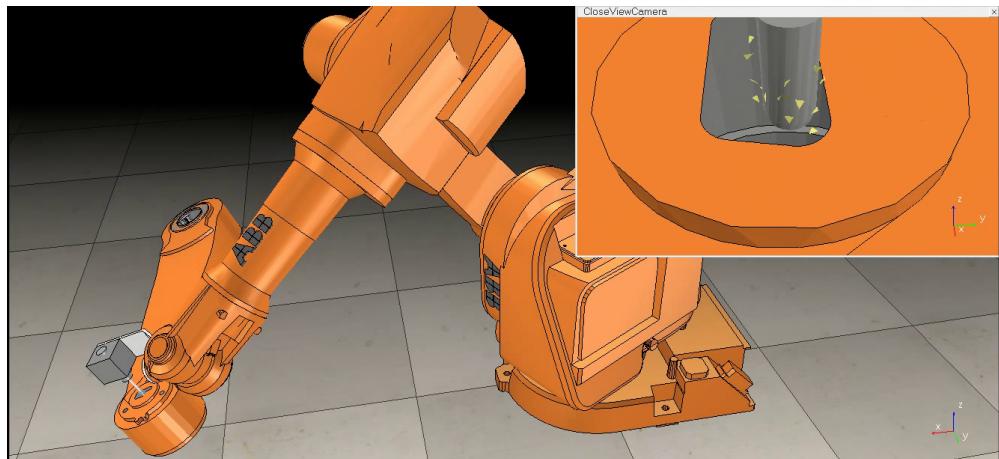


Figura 6.34. Braço robótico com fresa efetuando os cortes programados no material (cena “millingRobot.ttt”).

De maneira similar ao corte, o V-REP permite o depósito de materiais sobre os objetos da cena. Pode-se então, efetuar uma simulação da soldagem industrial usando braços robóticos³³. A Figura 6.35 mostra uma tela exemplo de uma operação de solda efetuada por um braço robótico no V-REP.

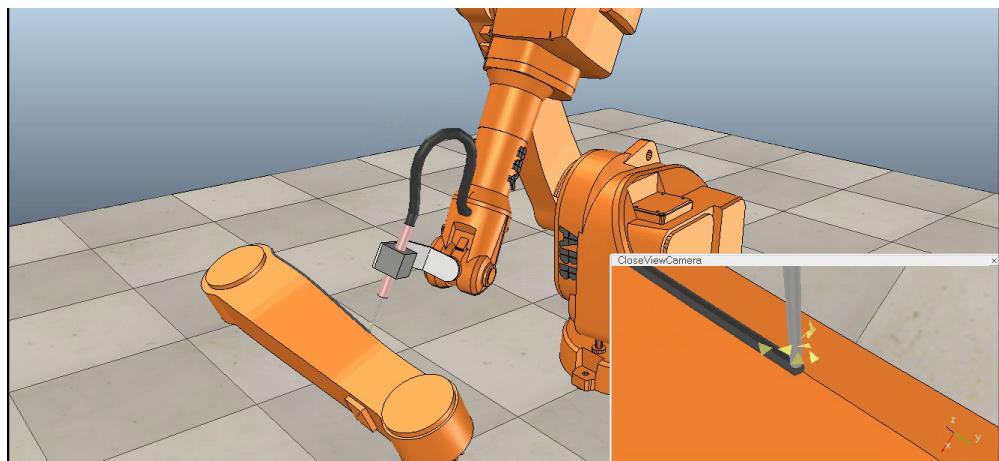


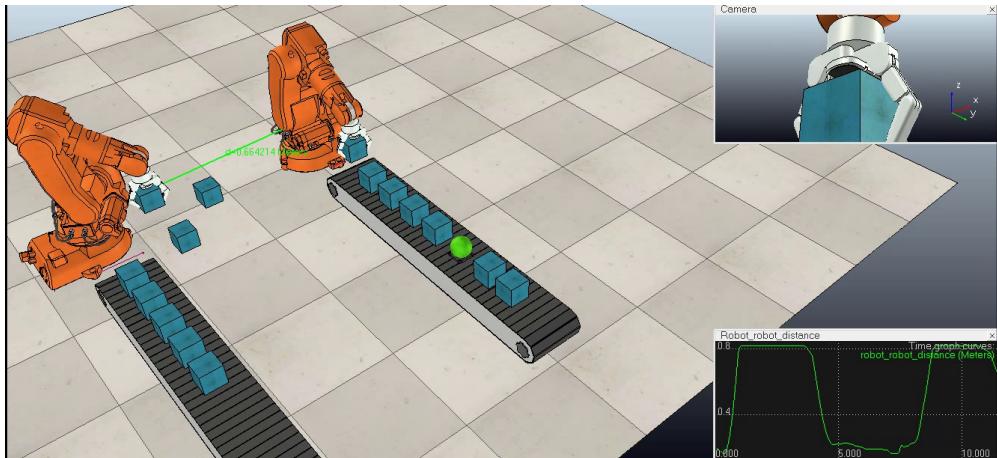
Figura 6.35. Braço robótico efetuado a solda em uma peça (cena “weldingRobot.ttt”).

Além de robôs e sensores, o V-REP possui alguns modelos já prontos que auxiliam na simulação de operações do cotidiano de parques fabris. Por exemplo, estão disponíveis modelos de esteiras que transportam mercadorias por fábricas. Atrelado a modelos de braços robóticos pode-se simular processos produtivos, ou mesmo, uma unidade de despacho de encomendas. Na Figura 6.36(a) é mostrada uma simulação com esteiras e robôs de base fixa³⁴ preparados para organizar caixas (despacho de mercado-

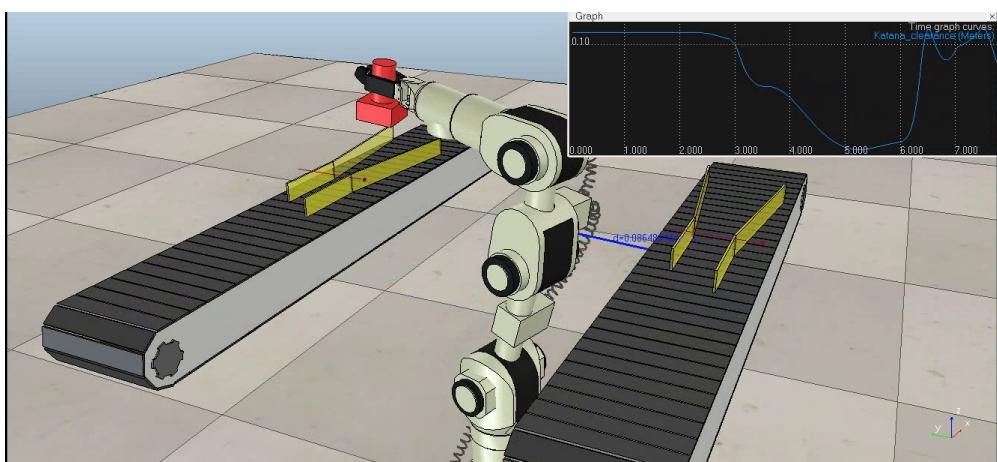
³³Vídeo exemplo do braço robótico soldador em V-REP que está disponível em <https://youtu.be/zvUt0mqfGpM>.

³⁴Vídeo exemplo do braço robótico e esteiras organizando caixas está disponível em <https://youtu.be/7-15euZsHfE>.

rias). Na Figura 6.36(b) é apresentado um exemplo de cena, onde, esteiras de montagem são realimentadas por um braço robótico³⁵.



(a) Esteiras para organização de despacho de caixas (cena “BarrettHandPickAndPlace.ttt”)



(b) Esteira de montagem (cena “katanaRobotWithCableSimulation.ttt”)

Figura 6.36. Exemplos de simulação de esteiras industriais.

Além de braços robóticos articulados convencionais, estão disponíveis no V-REP robôs paralelos (*parallel manipulators*) específicos para manipulação em ambientes industriais. Estes robôs permitem uma movimentação de mercadorias mais eficiente que braços robóticos. Esses robôs podem ser afixados sobre as esteiras, possibilitando a automação da retirada de objetos em uma linha de produção. A Figura 6.37 apresenta um exemplo de linha industrial³⁶, onde, as mercadorias são destinadas segundo sua cor e forma. Esta simulação emprega dois robôs manipuladores IRB-360 [ABB 2015b], uma esteira e uma câmera acoplada no início da esteira para a obtenção da posição, forma e cor das peças.

³⁵No endereço <https://youtu.be/QE4GC4L1ioI> pode-se ver o vídeo gerado da simulação de esteiras de montagens alimentadas por braço robótico.

³⁶Está disponível o vídeo da simulação da esteira e os robôs manipuladores no endereço <https://youtu.be/iXDJ-fSr9Uw>.

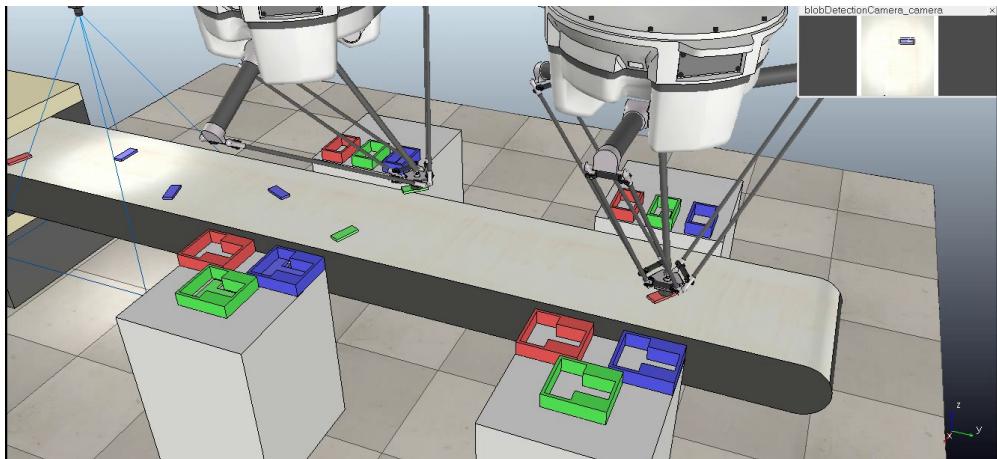


Figura 6.37. Exemplo de robôs manipuladores que atuam sobre uma esteira, fazendo a destinação de objetos segundo forma e cor (cena “blobDetectionWith-PickAndPlace.ttt”).

Além destes exemplos, é importante destacar que o V-REP também oferece modelos de diferentes tipos de veículos que podem ser acessados através do menu na opção *Load-Model* ou no *browser* de modelos. Estes modelos incluem robôs terrestres, na pasta *vehicles*, com modelos de carros (*manta.ttm*), de uma escavadeira (*excavator.ttm*), de uma motocicleta (*motorbike.ttm*) e por fim de um veículo com cinemática Ackermann na pasta *examples* (*SimpleAckermannsterring.ttm*). Além dos veículos terrestres também existem modelos de veículos aéreos, notadamente, de um helicóptero (pasta *vehicles*: *helicopter.ttm*) e de um quadcoptero (pasta *robots-mobile*: *Quadricopter.ttm*).

6.6. Robótica Móvel e Inteligente: Perspectivas futuras

Atualmente os robôs móveis e articulados já podem ser considerados “robôs inteligentes”, pois possuem a capacidade de: (i) perceber, decidir e agir de forma autônoma; (ii) conduzir veículos de modo autônomo; (iii) transportar e entregar mercadorias usando veículos aéreos; e inclusive, (iv) executar atividades humanas como caminhar, manipular e transportar objetos (robôs humanóides). Os robôs inteligentes tem sido aplicados na indústria (p.ex. braços industriais e manipuladores, AGVs), em ambientes domésticos (p.ex. robôs aspiradores de pó, cortadores de grama, limpadores de piscina), em atividades comerciais (p.ex. taxis robóticos, garçons automatizados), no entretenimento e na pesquisa (p.ex. competições robóticas, animais robóticos como cães e dinossauros, robôs para crianças e cuidadores de idosos), no ensino (p.ex. robôs usados no ensino de programação, robôs LEGO e NAO), em atividades que sejam arriscadas ou perigosas para seres humanos (p.ex. pulverização de agrotóxicos, colheita automatizada, exploração de outros planetas, mineração, busca e salvamento em desastres), no ar-terra-água (p.ex. robôs de exploração e monitoramento), e inclusive em atividades bélicas, com os Drones, sentinelas e outros robôs de combate. Podemos concluir que os robôs já estão plenamente disseminados em nossa sociedade atual e estarão cada vez mais presentes em nosso dia-a-dia, afinal quem de nós não possui um produto que foi manufaturado por um robô, como um *smartphone* ou mesmo um carro.

Neste contexto, é de grande importância que a sociedade atual esteja preparada para este futuro não muito distante, onde vamos conviver diariamente com robôs, e principalmente, precisamos preparar as gerações futuras para que estejam prontas para conviver, trabalhar e desenvolver novos projetos na área robótica. Além disto, devemos estar preparados para este futuro que nos aguarda, através da discussão sobre os aspectos legais e éticos que a robótica irá impor a sociedade, além de estar preparados para os impactos que esta automação e a “invasão do robôs” irá ter em nossa sociedade.

Muitas pessoas tem uma visão fantasiosa da robótica e do futuro, influenciadas pela ficção científica (que nem sempre reflete a realidade e uma expectativa futura mais realista). Por exemplo, um grande número de pessoas acredita que os robôs podem "roubar" nossos empregos e que serão os responsáveis pelo desemprego em massa no futuro, mas será que isto é verdade? Existem diversos estudos que demonstram que os robôs irão inclusive resolver enormes problemas dos humanos, como por exemplo, na produção de alimentos, onde a automação agrícola é indispensável para uma produção suficiente de alimentos para a atual população de nosso planeta. Outras tarefas como cuidar de idosos será muito em breve de grande importância, considerando o envelhecimento da população e o aumento da expectativa de vida. E afinal, quando aumentamos a produção e a produtividade (otimização da produção), isto não gera mais empregos? Esta é uma discussão importante, onde devemos buscar uma desmistificação da robótica, para que possamos aproveitar melhor os seus benefícios. Mas é claro que devemos nos preparar para os impactos que a automação pode trazer, pois quem sabe, algumas profissões podem mesmo ter que mudar, como a de motorista de táxi ou de garçom/entregador de mercadorias. De qualquer modo, não podemos ficar a margem desta grande revolução que é a inserção da robótica junto a sociedade moderna.

Em relação as questões legais e éticas, considerando a capacidade de um robô poder dirigir um carro sem a necessidade da supervisão constante de uma pessoa (indicada no item *(ii) acima*), um dos importantes questionamentos que surgem é: O que acontece em caso de acidentes? Quem é o responsável legal, uma vez que não havia uma pessoa conduzindo o veículo? Estes questionamentos demonstram claramente que o desenvolvimento de aplicações robóticas vai além das questões técnico-científicas, alcançando inclusive outras áreas como o Direito. Será necessária uma colaboração direta entre pesquisadores e experts em robótica, juntamente com juristas, a fim de estabelecer novas regras e leis para este tipo de situações. Neste caso em particular, acreditamos que também será necessário desenvolver equipamentos específicos, como caixas pretas que registrem tudo o que se passa dentro e fora (sensores) do carro, bem como serão necessárias leis específicas que obriguem a adoção de tais equipamentos tal como ocorre em aviões. Somente com tais equipamentos e com o registro detalhado da cena do acidente, poderemos determinar os responsáveis e penalizá-los legalmente, assim como será possível incrementar a segurança destes equipamentos. Isto demonstra quanto trabalho e a necessidade URGENTE de formar pessoal capacitado para tratar de questões como estas, seja no exterior ou seja em nosso país.

Considerando a capacidade de um robô aéreo aéreo indicada no item *(iii)*, onde um veículo aéreo não tripulado pode também cair, provocando danos materiais e pessoais (em alguns países drones acima de um determinado tamanho já são proibidos e/ou plenamente regulamentados em termos de seu uso). Constatase, assim como para os veículos

terrestres, que são necessários estudos e medidas em relação a segurança, mas também em relação a privacidade, uma vez que tais veículos (Drones, VANTs, UAVs) têm sido amplamente utilizados para realizar filmagens, invadindo casas e a privacidade das pessoas. E mais grave ainda, é que usualmente as pessoas não tem como impedir ou reagir, pois como evitar que um drone invada um espaço privado? Questões legais/éticas devem ser discutidas, e novamente, devem ser buscadas soluções não somente legais, mas também tecnológicas: quem sabe com o desenvolvimento de uma esquadra de uma esquadra com guardas-drones (*drone cops*) que irão “caçar” os drones ilegais. Isto envolve novamente a necessidade de formação e preparação de pessoal capacitado a desenvolver tais tecnologias e regulamentações.

Por fim, os robôs podem ser usados inclusive em atividades bélicas (e não somente os robôs humanóides, mas também os robôs terrestres, aéreos ou aquáticos). Isto pode ter consequências bastante perigosas para a sociedade, pois mesmo as guerras possuem suas regras, e fica aqui a pergunta: quem já viu um robô de guerra/ataque reconhecer e aceitar um pedido de rendição (bandeira branca) por parte de um humano?

Portanto, o conhecimento, o domínio da tecnologia, a regulamentação da robótica e a discussão de questões éticas/legais, são de grande importância na atualidade em relação a este tema. O Brasil, diante deste quadro, deve buscar um maior investimento e formação de pessoal na área de robótica, para que possamos estar preparados para estas situações com as quais muito em breve seremos confrontados. Em função disto, acreditamos que este mini-curso, através dos conceitos e ferramentas (V-REP) apresentados, tem uma grande relevância, por permitir um maior acesso as tecnologias e conhecimentos ligados a robótica, e assim, fomentar do desenvolvimento desta área estratégica no país.

Referências

- [ABB 2015a] ABB (2015a). Robô irb 140. Acessado: 19.03.2015.
- [ABB 2015b] ABB (2015b). Robô irb 360. Acessado: 19.03.2015.
- [Adept MobileRobots 2015] Adept MobileRobots, I. (2015). Robô pioneer p3dx. Acessado: 19.03.2015.
- [Aldebaran Robotics 2015] Aldebaran Robotics, I. (2015). Robô humanoíde nao. Acessado: 19.03.2015.
- [Arkin 1989] Arkin, R. C. (1989). Towards the unification of navigational planning and reactive control. In *In AAAI Spring Symposium on Robot Navigation*, pages 1–5.
- [Arkin and Balch 1997] Arkin, R. C. and Balch, T. (1997). Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189.
- [Bradski and Kaehler 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc.
- [Cook et al. 2014] Cook, D., Vardy, A., and Lewis, R. (2014). A survey of auv and robot simulators for multi-vehicle operations. In *Autonomous Underwater Vehicles (AUV), 2014 IEEE/OES*, pages 1–8.

- [Coppelia Robotics 2015] Coppelia Robotics, G. (2015). Virtual robot experimentation platform - user manual. <http://www.coppeliarobotics.com/helpFiles/>. Acessado em: 19.03.2015.
- [Corke 2011] Corke, P. I. (2011). *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer.
- [Correa 2013] Correa, D. S. O. (2013). *Navegação autônoma de robôs móveis e detecção de intrusos em ambientes internos utilizando sensores 2D e 3D*. USP ICMC - Dissertação de Mestrado, Sao Carlos, SP, Brasil.
- [Craighead et al. 2008] Craighead, J., Gutierrez, R., Burke, J., and Murphy, R. (2008). Validating the search and rescue game environment as a robot simulator by performing a simulated anomaly detection task. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2289–2295.
- [Cytron Technologies 2015] Cytron Technologies, I. (2015). Sonar hc-sr04. <http://cytron.com.my/p-sn-hc-sr04>. Acessado em: 19.03.2015.
- [Department 2014] Department, I. S. (2014). World robotics 2014 - industrial robots and service robots (executive summary wr-2014). http://www.worldrobotics.org/uploads/media/Executive_Summary_WR_2014_02.pdf. Acessado em: 08.05.2015.
- [Dudek and Jenkin 2000] Dudek, G. and Jenkin, M. (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge, UK.
- [E. Rohmer 2013] E. Rohmer, S. P. N. Singh, M. F. (2013). V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.
- [Echeverria et al. 2011] Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51.
- [Evollve 2015] Evollve, I. (2015). Ozobot. <http://www.ozobot.com/>. Acessado em: 19.03.2015.
- [Fernandes et al. 2014] Fernandes, L. C., Souza, J. R., Pessin, G., Shinzato, P. Y., Sales, D., Mendes, C., Prado, M., Klaser, R., Magalhães, A. C., Hata, A., et al. (2014). Carina intelligent robotic car: Architectural design and applications. *Journal of Systems Architecture*, 60(4):372–392.
- [Freescale Semiconductor 2015] Freescale Semiconductor, I. (2015). Acelerômetro mma7361l. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA7361LC&tid=vanRED. Acessado em: 19.03.2015.
- [Gates 2007] Gates, B. (2007). A robot in every home. *Scientific American*, 296(1):58–65.

- [Gerkey et al. 2003] Gerkey, B., Vaughan, R. T., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323.
- [Grassi Jr. and Okamoto Jr. 2014] Grassi Jr., V. and Okamoto Jr., J. (2014). Arquiteturas de controle: tipos e conceitos. In Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D., editors, *Robótica Móvel*, chapter 4, pages 47–60. Editora LTC, Rio de Janeiro.
- [Guizzo 2012] Guizzo, E. (2012). Amazon acquires kiva systems for \$775 million. <http://spectrum.ieee.org/automaton/robotics/industrial-robots/amazon-acquires-kiva-systems-for-775-million>. Acessado em: 19.03.2015.
- [Harris and Conrad 2011] Harris, A. and Conrad, J. M. (2011). Survey of popular robotics simulators, frameworks, and toolkits. In *Southeastcon, 2011 Proceedings of IEEE*, pages 243–249.
- [HiBot 2015] HiBot (2015). Robô serpente acm-r5. Acessado: 19.03.2015.
- [Hokuyo Automatic 2015] Hokuyo Automatic, C. (2015). Urg-04lx-ug01. https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html. Acessado em: 19.03.2015.
- [InvenSense 2015] InvenSense, I. (2015). Imu mma73611. <http://www.invensense.com/mems/gyro/mpu6050.html>. Acessado em: 19.03.2015.
- [iRobot 2015] iRobot (2015). Roomba. <http://www.irobot.com/For-the-Home/>. Acessado em: 19.03.2015.
- [Iyengar and Elfes 1991] Iyengar, S. S. and Elfes, A. (1991). *Autonomous Mobile Robots: Perception, mapping, and navigation*. Autonomous Mobile Robots. IEEE Computer Society Press.
- [Jung et al. 2005] Jung, C. R., Osorio, F. S., Kelber, C., and Heinen, F. (2005). Computação embarcada: Projeto e implementação de veículos autônomos inteligente. In *Anais do CSBC'05 XXIV Jornada de Atualização em Informática (JAI)*, chapter 4, pages 1358–1406. SBC, São Leopoldo, RS.
- [Kiva Systems 2015] Kiva Systems, L. (2015). Automated guided vehicles. <http://www.kivasytems.com/>. Acessado em: 19.03.2015.
- [Koenig and Howard 2004] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154.
- [Landroid 2015] Landroid (2015). Wg794e. <http://www.worxlandroid.com/>. Acessado em: 19.03.2015.

- [Mataric 1992] Mataric, M. (1992). Behavior-based control: Main properties and implications. In *In Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pages 46–54.
- [Mataric 2014] Mataric, M. (2014). *Introdução a Robótica*. Blucher/UNESP Editora, São Paulo, SP, Brasil, 1ed(traduzido) edition.
- [Metta et al. 2006] Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48.
- [Murphy 2000] Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition.
- [NASA 2015] NASA (2015). Robonaut r2. <http://robonaut.jsc.nasa.gov/>. Acessado em: 19.03.2015.
- [Oxford Mobile Robotics 2015] Oxford Mobile Robotics, G. (2015). Moos. <http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage>. Acessado em: 28.04.2015.
- [Pocolibs 2015] Pocolibs (2015). <https://www.openrobots.org/wiki/pocolibs/>. Acessado em: 28.04.2015.
- [Point Grey Research 2015] Point Grey Research, I. (2015). Point grey chameleon. <http://www.ptgrey.com/>. Acessado em: 19.03.2015.
- [PUC-Rio 2015] PUC-Rio, G. (2015). The programming language lua. <http://www.lua.org/>. Acessado em: 19.03.2015.
- [Quigley et al. 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.
- [Romero et al. 2014a] Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D. (2014a). Introdução à Robótica Móvel. In *Robótica Móvel*, chapter 1, pages 1–12. Editora LTC, Rio de Janeiro.
- [Romero et al. 2014b] Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D. (2014b). Sensores e Atuadores. In *Robótica Móvel*, chapter 2, pages 13–25. Editora LTC, Rio de Janeiro.
- [SICK 2015] SICK, A. (2015). Lms 200. <http://www.sick.com/>. Acessado em: 19.03.2015.
- [Siegwart et al. 2011] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition.
- [Staranowicz and Mariottini 2011] Staranowicz, A. and Mariottini, G. L. (2011). A survey and comparison of commercial and open-source robotic simulator software. In *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA ’11, pages 56:1–56:8, New York, NY, USA. ACM.

- [Vaughan et al. 2003] Vaughan, R. T., Gerkey, B. P., and Howard, A. (2003). On device abstractions for portable, reusable robot code. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2421–2427.
- [Velodyne LiDAR 2015] Velodyne LiDAR, I. (2015). Velodyne hdl-32e. <http://velodynelidar.com/>. Acessado em: 19.03.2015.
- [Vieira and Roqueiro 2014] Vieira, R. d. S. and Roqueiro, N. (2014). Cinemática e dinâmica de robôs móveis. In Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D., editors, *Robótica Móvel*, chapter 3, pages 26–46. Editora LTC, Rio de Janeiro.
- [Wolf et al. 2009] Wolf, D., Simoes, E., Osorio, F., and Trindade Junior, O. (2009). Robótica inteligente: Da simulação as aplicações no mundo real. In *Anais do CSBC'09 XXIV Jornada de Atualização em Informática (JAI)*, chapter 6, pages 279–330. SBC, Bento Gonçalves, RS.