Fundamentos matemáticos e computacionais de problemas de roteamento de veículos*

Pedro Henrique Centenaro†

Orientador: Luiz Rafael dos Santos

Resumo. Este trabalho visa estabelecer os princípios matemáticos necessários ao entendimento de problemas de roteamento de veículos (PRVs), capacitados e com janelas de tempo, e os modelos propostos na literatura para resolvê-los. Como motivação, são apresentados fatos e tendências sobre urbanização e transporte público, bem como as vantagens e desafios da troca de veículos a combustão por veículos elétricos, cujas peculiaridades influenciam a modelagem dos PRVs. Abordam-se conceitos de grafos, otimização linear e inteira, bem como as implementações computacionais do método simplex e do problema do caixeiro viajante (PCV), e comparam-se os desempenhos de três *solvers* e formulações do PCV. Por fim, três modelos de PRVs são apresentados e comparados.

*Este trabalho foi financiado pela Propesq e PIBIC-CNPq/UFSC

†Acadêmico do curso de Bacharelado em Engenharia de Controle, Automação e Computação da UFSC/Campus Blumenau.

Sumário

1. Introdução **4**

2. Urbanização e transporte público **6**

2.1. Vantagens e tendências do transporte público 6

2.2. Sobre ônibus elétricos 8

2.3. O desafio do espraiamento no transporte público brasileiro 12

3. Modelagem matemática para roteirização **15**

3.1. Grafos e dígrafos simples 15

3.2. Conexões entre vértices 17

3.3. O problema do caixeiro viajante 18

3.4. Problemas de otimização 22

3.5. Poliedros e convexidade 24

3.6. Ponto extremo e vértice 26

3.7. Solução básica factível 28

3.8. Equivalências 32

3.9. Resumo do método simplex 33

4. Considerações computacionais para roteirização **35**

4.1. Metodologia de modelagem em ambiente computacional 35

4.2. Técnicas de otimização inteira 38

4.2.1. Técnicas exatas 38

4.2.2. Técnicas heurísticas 39

4.3. Comparações computacionais 41

4.3.1. Comparação de formulações 41

4.3.2. Comparação de solvers 42

5. Problemas de roteamento de veículos com janelas de tempo **47**

5.1. Modelo genérico 47

5.2. Modelo escolar com rotas pré-determinadas 49

5.3. Modelo de frota mista 51

5.4. Comparação de modelos 53

6. Conclusão **55**

Referências **56**

A. O método simplex **63**
A.1. Teoremas fundamentais para o método simplex . . . 63
A.2. Considerações iniciais . . . 65
A.3. Representação de um problema em forma canônica . . . 66
A.4. Variáveis básicas e não-básicas . . . 68
A.5. Mecanismo iterativo de redução do valor objetivo . . . 70
A.6. Uma formulação inicial do simplex . . . 73
A.7. O simplex tableau . . . 76
A.7.1. Construção do tableau inicial . . . 77
A.7.2. Cálculo de nova iteração por pivoteamento . . . 78
A.8. Obtenção da solução básica inicial pelo método de duas fases . . . 83
A.8.1. O que constitui uma solução básica inicial aceitável . . . 83
A.8.2. Obtenção da submatriz identidade por variáveis artificiais . . . 84
A.8.3. O problema de primeira fase . . . 86
A.9. Solucionando problemas envolvendo degeneração . . . 87
A.9.1. Degeneração ao fim da primeira fase . . . 87
A.9.2. Caso geral de degeneração . . . 88
A.9.3. O método lexicográfico . . . 90
A.10. Resolução de problemas de otimização linear com o Caique.jl . . . 94

B. Códigos de roteirização **96**
B.1. Implementações do PCV . . . 96
B.1.1. Formulação DFJ . . . 96
B.1.2. Formulação DFJ com lazy constraints . . . 96
B.1.3. Formulação MTZ . . . 99
B.1.4. Formulação GG . . . 99
B.2. Implementação do PRVJT genérico . . . 99

1. Introdução

Este trabalho foi desenvolvido, de abril de 2022 a agosto de 2023, pelo discente Pedro Henrique Centenaro, estudante de Engenharia de Controle, Automação e Computação na Universidade Federal de Santa Catarina (UFSC), Campus Blumenau. O professor Luiz Rafael dos Santos atuou como orientador do projeto, e foi fornecido ao estudante acesso ao Laboratório de Matemática Aplicada e Computacional da UFSC, o LABMAC

A roteirização de veículos é um processo logístico fundamental para empresas. Entre suas aplicações, podem ser citadas a coleta de materiais recicláveis [4], coleta de lixo [54], distribuição de alimentos perecíveis [3], roteirização de atendimento médico [44] e roteirização de ônibus [39]. Os modelos matemáticos aplicados à resolução de tais problemas têm formulações variadas, cada qual dependendo da complexidade da análise realizada pelos autores.

O problema mais simples de roteirização é o do caixeiro viajante (PCV), em que um único veículo deve atender determinado número de pontos de modo a minimizar seus gastos com deslocamentos [37, 63, 69]. Quando se considera uma frota, tem-se um problema de roteamento de veículos (PRV) [38]. Neste caso, é possível considerar uma quantidade muito maior de obstáculos à roteirização, como capacidades dos veículos, janelas de tempo de atendimento, múltiplos depósitos, frotas mistas e outros mais [73, 76].

Com o crescimento dos modelos dos problemas de roteirização, aumenta a complexidade computacional para resolvê-los. Não é incomum que se discutam soluções para PCVs relativamente pequenos que encontrem a solução ótima do problema; no entanto, para os PRVs, a prática exige que soluções quase-ótimas sejam consideradas [48]. Ainda assim, tanto o PCV quanto os PRVs são problemas NP-difíceis [78]. Por exemplo, a resolução de um PCV de 85900 pontos necessitou do uso de 96 estações de trabalho e um poder computacional que equivale a 139 anos de funcionamento de uma única CPU [6].

Tendo em vista a complexidade teórica e prática dos PRVs, o objetivo geral deste trabalho é explorar e explicar as principais características que contribuem para as dificuldades de implementação destes problemas. Já os objetivos específicos são:

- Estudar as teorias dos problemas de roteamento de veículos capacitado e com janela de tempo, incluindo aplicações;
- Modelar PRVJT escrevendo as equações relacionadas e implementar o modelo na linguagem de modelagem JuMP;

- Fazer testes computacionais com diversos *solvers*, em particular que utilizem técnicas exatas e heurísticas/meta-heurísticas;
- Reportar os resultados em formato de relatório técnico e pôster em eventos científicos.

Visando alcançar tais objetivos, os princípios matemáticos e computacionais necessários para a compreensão e implementação de modelos de roteamento foram concatenados neste relatório. A ideia é que o público-alvo seja capaz de ler o trabalho por completo, chegando ao fim com a capacidade de interpretar modelos complexos de PRVs e identificar as vantagens e desvantagens computacionais de cada tipo de formulação.

Para realizar a leitura deste documento, recomenda-se um conhecimento básico sobre demonstrações e notações matemáticas, além de álgebra linear. Na Seção 2, o transporte público urbano, seus desafios e tendências futuras são apresentados como motivadores para o estudo de PRVs. Na Seção 3, apresenta-se o básico de grafos, seguido de uma definição matemática formal do PCV. Pelo resto da seção, discutem-se objetos da álgebra linear necessários para compreender a ideia por trás do método simplex, muito utilizado para resolver problemas de otimização. Os mecanismos por trás deste método são discutidos em detalhes no Apêndice A, incluindo pseudocódigo e implementação [20] (ver o pacote `Caique.jl`).

A Seção 4 lida com os aspectos computacionais de implementação e resolução de modelos. São comparados *solvers*[1] e formulações do PCV, levando em conta a teoria desenvolvida até o momento. Detalhes de implementação podem ser encontrados no Apêndice B.1. Na Seção 5, são apresentados e descritos alguns modelos de PRVJTs. Similaridades e diferenças notáveis entre os modelos são comparadas, realçando o impacto que as considerações específicas de cada um têm sobre seu formato matemático e suas características computacionais. Finalmente, na Seção 6, os objetivos deste trabalho são revisitados de modo a avaliar se e como foram cumpridos.

[1] Como é detalhado em seções posteriores, os *solvers* são ferramentas computacionais robustas que tentam resolver problemas de otimização por meio de várias técnicas exatas e heurísticas.

2. Urbanização e transporte público

O transporte público é uma necessidade ubíqua no Brasil. Particularmente nos centros urbanos, é comum que existam dificuldades na implementação e manutenção de frotas, devido à alta densidade populacional dos centros e à marginalização de setores da população, que diuturnamente precisam viajar por longos trajetos para garantirem seu sustento. Tendo em vista a grande importância deste assunto, nesta seção, são abordados fatos e tendências do transporte público, no Brasil e no mundo, que servem como inspiração para o estudo da modelagem e resolução de problemas de roteamento de veículos.

2.1. Vantagens e tendências do transporte público

Litman [52], em pesquisa conduzida nos Estados Unidos, descreve vários benefícios do transporte público para seus usuários. Alguns dos mais importantes que podemos elencar são:

- **Locomoção mais variada:** Quanto melhor a estrutura do transporte público de uma região, aliada à promoção de bicicletas, caminhadas e transportes alternativos, menor é o uso de automóveis particulares por parte da população. Verifica-se ainda que a média de automóveis particulares, por residência, em regiões com transporte público de alta qualidade, pode chegar a metade do que se observa em regiões com pouco investimento em transporte público.

- **Aumento do tempo de exercício físico:** Usuários de transporte público ou alternativo precisam andar pelo menos duas vezes mais para chegar aos seus destinos diários. Esta vantagem de tempo gasto com exercícios se mantém, visto que o tempo gasto com exercícios recreativos é aproximadamente o mesmo para usuários de transporte público ou alternativo e usuários de automóveis privados.

- **Declínio do número de fatalidades no trânsito:** Quando bem implementado e largamente utilizado, o transporte público incentiva mais cuidado por parte de motoristas e da legislação, pois mais pessoas passam a andar a pé ou de bicicleta. Como mostram estatísticas internacionais, comunidades em que o transporte público é bem integrado à vida das pessoas têm consideravelmente menos fatalidades no trânsito.

- **Redução da poluição veicular:** Ônibus elétricos e ônibus movidos a combustão mais recentes produzem cada vez menos gases nocivos, conforme evoluem as

regulamentações sobre emissões. Além disso, como estes veículos podem transportar dezenas de pessoas e desincentivam o uso de veículos particulares, a poluição também é reduzida graças à redução do número de veículos nas ruas.

- **Integração comunitária:** Em regiões com transporte público de qualidade, a população tem acesso facilitado a serviços essenciais (saúde, educação, emprego, víveres, etc.), o que significa que as pessoas em geral terão melhores condições de saúde, mais oportunidades econômicas, menores gastos com transporte e manutenção de veículos particulares e melhores condições de saúde mental. Todas estas questões são também importantes para a administração da cidade, pois o acesso facilitado a atendimento médico permite que as pessoas se consultem antes que seus problemas de saúde se agravem. Da mesma maneira, o acesso facilitado a educação e emprego significa que mais pessoas terão acesso a maiores graus de instrução, o que pode aumentar a diversidade de profissionais do mercado da região e beneficiar a economia.

Ademais, Schwab [67] apresenta considerações importantes sobre as tendências de desenvolvimento das cidades. De acordo com relatório de pesquisa do Fórum Econômico Mundial [57], que ouviu 816 executivos de grandes empresas de tecnologia e inovação, a expectativa, em 2015[2], de 64% dos entrevistados era que até 2025 surja a primeira cidade com mais de 50 mil habitantes e sem semáforos. Diz também Schwab que

> [...] os avanços tecnológicos de sensores, sistemas óticos, processadores embutidos, maior segurança para os pedestres e para o transporte não motorizado levarão à maior adoção do transporte público, redução dos congestionamentos e da poluição, melhor saúde e trajetos mais rápidos, mais previsíveis e menos caros.

Schwab também argumenta que as cidades que mais rapidamente conseguirem integrar novas tecnologias aos seus espaços públicos atrairão o maior número de pessoas qualificadas para o seu desenvolvimento. Ele acrescenta:

> [...] estudos mostram que o aumento das áreas verdes de uma cidade em 10% poderia compensar o aumento da temperatura causado pelas mudanças climáticas [...].

[2]Talvez esta expectativa tenha sido atrasada desde então, devido à pandemia de Covid-19.

Segundo Bursztyn e Eiró [14], a preocupação com as mudanças climáticas e seus efeitos é maior entre pessoas com maior grau de escolaridade. Logo, as tecnologias verdes (particularmente, como argumentaremos, os ônibus elétricos) devem ser foco de investimento por parte das cidades que desejarem atrair estes demográficos.

Um estudo de Lima, Silva e Albuquerque Neto [71] aponta iniciativas de adoção do transporte por ônibus elétricos em cidades como Amsterdã, Cidade do Cabo, Copenhague, Hamburgo, Los Angeles, Nova Iorque, Oslo, Rugao e São Francisco. O objetivo destas cidades é substituir completamente os ônibus a combustão pelos elétricos.

Em suma, existe uma tendência entre os países mais desenvolvidos de investir nas novas tecnologias elencadas pelo Fórum Econômico Mundial [57], sendo de particular interesse para esta análise as que dizem respeito ao meio ambiente. Estas tecnologias são importantes pela redução dos impactos ambientais dos meios urbanos e pela crescente preocupação da população com seus efeitos no planeta. Como argumentamos a seguir, na Seção 2.2, é importante que as cidades brasileiras tentem seguir estas tendências adicionando ônibus elétricos a suas frotas e retirando sempre que possível os ônibus a combustão de circulação.

2.2. Sobre ônibus elétricos

Para entender os investimentos em ônibus elétricos realizados por grandes cidades, precisamos elencar seus benefícios em relação aos ônibus a combustão. O trabalho de Lima, Silva e Albuquerque Neto [71] é bastante esclarecedor a este respeito. Os autores discorrem sobre a existência de dois tipos de ônibus elétricos, os totalmente elétricos e os híbridos, apontando suas principais características. Em seguida, discutem os benefícios ambientais da implementação de tais veículos no transporte público. Por fim, os autores abordam algumas das principais dificuldades relacionadas à implementação destes veículos.

Ônibus totalmente elétricos

Este tipo de ônibus funciona com base em baterias recarregáveis. Por este motivo, gera consideravelmente menos partículas do que modelos a combustão, além de suas emissões de gases de efeito estufa serem nulas. Um benefício de longo prazo é que este tipo de ônibus é mecanicamente mais simples, sendo mais barata a sua manutenção. Ônibus totalmente elétricos podem ser movidos por vários tipos de motores elétricos, que têm a vantagem de fornecerem maior aceleração ao veículo do que motores convencionais. Devido às suas características, ônibus totalmente elétricos têm aproximadamente três

vezes a eficiência dos ônibus a combustão (59% a 62% de eficiência contra 17% a 21% de eficiência, respectivamente).

As baterias de ônibus totalmente elétricos têm um tempo de vida útil estimado entre 8 e 10 anos, podendo ser utilizadas para alimentar outros tipos de serviço após este período. Atualmente, as baterias mais utilizadas são as de íons de lítio, graças à sua eficiência. No entanto, sua reciclagem é muito custosa. É possível que no futuro estas baterias percam sua posição dominante no mercado para as de níquel-hidreto metálico, que têm grande vida útil mas, atualmente, têm alto custo e desvantagens físicas comparadas as de íons de lítio.

Uma desvantagem dos ônibus totalmente elétricos em relação aos de combustão é que o projeto dos sistemas auxiliares (como portas, direção hidráulica e compressor do ar condicionado) é mais complicado, visto que cada sistema pode precisar do seu próprio motor em configurações específicas. Desta forma, o projeto de controle individual para cada um destes motores acarreta custos adicionais com componentes, o que pode deixar estes veículos mais complexos.

Ônibus híbridos

Ônibus híbridos misturam aspectos de ônibus totalmente elétricos com ônibus a combustão, sendo mais eficientes que os modelos a combustão e tendo a possibilidade de serem utilizados como fonte de energia para outros veículos. Ademais, seus componentes têm tamanho reduzido com o uso inteligente das baterias.

Estes veículos são hibridizados por meio de extensores de autonomia, definidos pelos autores como

> [...] sistemas embarcados capazes de gerar energia durante a operação, recarregando as baterias e consequentemente estendendo a autonomia do veículo.

Existem vários tipos de extensores de autonomia, sendo os principais as pilhas a combustível, grupo motor-gerador, motores a combustão interna e supercapacitores. Em geral, estes componentes permitem a recuperação ou maior eficiência energética, garantindo mais eficiência de operação. Vale prestar atenção à seguinte observação dos autores:

> [...] em percursos com baixa necessidade de frenagens, como rodovias, as vantagens relativas à recuperação de energia diminuem a eficiência do veículo híbrido. Dessa forma, tais veículos se aproveitam melhor deste sistema em percursos com um maior número de frenagens, que é o caso do transporte urbano.

Os veículos híbridos podem ser classificados como: em série, caso no qual são movidos por motor elétrico alimentado pelas baterias ou pelos extensores de autonomia; em paralelo, podendo ser tracionados por motor elétrico ou a combustão, juntos ou individualmente; e em série-paralelo, unindo características dos dois.

Benefícios ambientais e econômicos

Os ônibus elétricos são muito mais eficientes do que os modelos a combustão no que se refere a emissão de substâncias nocivas ao meio-ambiente. Um problema que pode surgir com a implementação deste tipo de ônibus diz respeito ao trajeto pelo qual passa a energia até abastecê-lo. Se esta energia não for produzida de maneira limpa, pode ser enganosa a noção de que a implementação de ônibus elétricos não acarreta a produção de poluição. Felizmente, no Brasil, 74% da energia elétrica provém de fontes limpas e renováveis. Além disso, estas fontes, em geral, ficam a grande distância da população, o que permite maior facilidade na captura de quaisquer materiais particulados que o processo de fornecimento de energia aos ônibus elétricos possa gerar, sem danos à população.

No que se refere à redução da emissão de gás carbônico, os autores observam que

> [...] [no Brasil,] a substituição dos ônibus e micro-ônibus por modelos elétricos, considerando toda a geração de energia para carregamento desses ônibus, garantiria uma redução de 17,44 milhões de toneladas de CO_2 (91,4%) para o ano de 2012.

Quanto aos materiais particulados, os autores esclarecem que

> a produção de energia elétrica para alimentar a frota de micro-ônibus e ônibus urbanos elétricos emitiria 58,9% mais material particulado do que a própria combustão do diesel [71].

No entanto, como já foi esclarecido, o Brasil tem as condições geográficas necessárias para lidar com estes materiais particulados eficientemente.

Os ônibus elétricos – e os híbridos, quando operando somente com base em motor elétrico – também ocasionam consideravelmente menos poluição sonora do que ônibus a combustão, tornando os espaços urbanos mais agradáveis e saudáveis.

Em geral, ônibus elétricos ainda requerem investimentos mais caros do que ônibus a combustão. No entanto, verifica-se que, escolhendo baterias apropriadas para a distância percorrida diariamente por cada ônibus, após atingirem determinada quilometragem, os ônibus elétricos passam a ser mais baratos do que os modelos a combustão. Além disso, as

tendências internacionais indicam que os aumentos no preço do petróleo serão maiores do que no preço da eletricidade, e espera-se que os conjuntos de baterias dos ônibus elétricos se tornem consideravelmente mais baratos até 2030, ano em que ônibus elétricos devem custar tanto ou menos que ônibus a combustão[3].

Dificuldades de implementação

Em geral, como os ônibus elétricos ainda são mais caros do que ônibus a combustão, são vistos como investimentos mais arriscados pelas operadoras de ônibus. Muitos dos ônibus elétricos adquiridos por países europeus vieram de subsídios federais, o que não constitui uma forma viável de continuar adquirindo estes veículos a longo prazo. Existe também o fato de ônibus a combustão já serem consagrados e serem impulsionados por indústrias e fabricantes já muito fortalecidos em cenários nacionais, especialmente em países em desenvolvimento.

A transição para modelos elétricos também é confrontada por problemas de padronização de infraestruturas de carregamento, apesar de haver avanços neste sentido proporcionados por acordos entre fabricantes de veículos elétricos. Existem também receios quanto à possibilidade de apagões afetarem as operações de transporte público, e há ainda a possibilidade de muitas cidades estarem esperando os preços de ônibus elétricos abaixarem antes de iniciarem seus investimentos neste tipo de tecnologia.

Apesar destas dificuldades, os benefícios dos ônibus elétricos têm peso considerável pelos motivos apresentados até aqui. Na conclusão de seu trabalho, Lima, Silva e Albuquerque Neto fazem a seguinte observação a respeito do futuro desta categoria de veículos:

> [...] Além dos novos arranjos de financiamento, é necessário se pensar em novos desenhos de contrato com alocação de risco adequada e que garanta que o risco da tecnologia seja alocado a quem tem maior capacidade de suportá-lo. É possível que novos *players* entrem no mercado de transporte urbano, como fabricantes de veículos oferecendo *leasing* e manutenção dos ônibus, e *utilities* de energia oferecendo infraestrutura de carregamento.

Assim, é possível esperar que ônibus elétricos se tornem cada vez mais atraentes e viáveis conforme seu mercado se desenvolve. Feita esta análise, é importante considerar desafios especificamente brasileiros que a adesão a este tipo de veículo pode encontrar.

[3] Estas expectativas são da Bloomberg New Energy Finance [30], e foram publicadas no ano de 2018. Novamente, é importante ressaltar a possibilidade de a pandemia de Covid-19 ter afetado as expectativas.

2.3. O desafio do espraiamento no transporte público brasileiro

Dados do Ipea [18] mostram que cerca de 85% da população brasileira vive hoje em centros urbanos, sendo 36 cidades com mais de 500 mil habitantes e um restante de quarenta regiões metropolitanas que abrigam uma população total de mais de 80 milhões de habitantes. Os dados indicam que a intensa industrialização brasileira da década de 1940 em diante, que estimulou vários êxodos rurais[4], não foi acompanhada por uma urbanização de similar velocidade.

Uma consequência da urbanização mal-elaborada do Brasil é o espraiamento [58], condição na qual os grandes centros urbanos, em vez de abrigarem todas as camadas da população, tornam-se excludentes. Os afetados por esta exclusão são os mais pobres, que precisam se estabelecer em regiões periféricas para terem onde morar. Como consequência disso, estas pessoas têm consideravelmente mais dificuldade de acesso a serviços públicos essenciais, que ficam concentrados nos centros urbanos. Como o transporte público está incluso na lista de serviços que são precarizados para estas regiões, muitas pessoas acabam recorrendo ao uso de veículos particulares, o que gera mais congestionamentos e acidentes [18].

O aumento do número de acidentes, em particular, se deve não apenas ao aumento do número de veículos em circulação, mas às próprias características do espraiamento. Litman [52] explica que isso se deve ao fato de os trajetos longos necessários para se chegar aos centros urbanos estimularem uma direção menos segura, com menos atenção por parte dos motoristas e mais velocidade nas pistas. Consequentemente, embora estes longos trajetos tendam a ter menos acidentes, os acidentes que acontecem tendem a ser mais graves, ou mesmo fatais. Isso contrasta com os benefícios do transporte público identificados na Seção 2.1, um dos quais é a redução da severidade dos acidentes em centros urbanos bem-planejados, apesar de concentrarem maior número de acidentes.

O exemplo de espraiamento de Araraquara

Borchers e Figueirôa-Ferreira [11] descrevem um caso de espraiamento na cidade de Araraquara, São Paulo. Em 1959, foi fundada a Companhia Troleibus Araraquara (CTA), uma companhia pública que oferecia serviço de transporte por trólebus para a população. Os trólebus eram ônibus elétricos alimentados por um sistema de cabos suspensos sobre as ruas da cidade. A princípio, a companhia tinha um centro urbano bem-delimitado

[4]Por êxodo rural, entende-se o deslocamento em massa de pessoas das zonas rurais do país para centros urbanos. Por exemplo, o êxodo rural que ocorreu no Brasil entre 1950 e 1995, onde cerca de 50 milhões de habitantes realizaram tal deslocamento [1].

para atender, sendo capaz de servir satisfatoriamente à população. No entanto, na década de 1970, começaram a surgir os primeiros grandes obstáculos ao serviço, na forma de loteamentos espraiados. Como os trólebus tinham sua mobilidade limitada por cabos, argumentou-se que seria mais difícil e custoso implementar novas linhas do serviço que atendessem os espraiamentos. Além disso, as ruas dos bairros espraiados não eram asfaltadas, o que trazia ainda mais problemas para a implementação dos trólebus. Assim, uma solução privada por ônibus a combustão apareceu na cidade, conseguindo atender as regiões mais espraiadas. Em 1999, a própria CTA abandonou os trólebus em prol de uma frota baseada totalmente em combustão, devido à maior simplicidade de atendimento às regiões espraiadas.

O objetivo deste exemplo é identificar por que o espraiamento pode acontecer. A conclusão à qual os autores chegaram, neste caso, é de que uma lógica *neoliberal* afetou o desenvolvimento urbano da cidade. Os autores definem o neoliberalismo como

> [...] uma doutrina econômico-política e filósofo-cultural que instrumentaliza de forma coercitiva e coesiva forças de acumulação por espoliação através do aparelhamento do Estado, do controle dos meios de produção – de bens, serviços, do espaço e do próprio ser humano – e do domínio ou subjugo de todos aqueles cuja existência serve apenas para a produção de mais-valia [...].

Os autores definem *urbanismo neoliberal* como

> [...] a específica aplicação dessa doutrina [neoliberal] de destruição criativa no planejamento e gestão do espaço urbano, de seus habitantes e aspectos econômicos, para reconfigurar a organização territorial e assim suscitar novas formas de produção desigual do espaço urbano e acumulação de capital [...].

Segundo o estudo, Araraquara era uma cidade marcada por grandes propriedades rurais. Com os êxodos rurais, os grandes proprietários de terras incentivaram a urbanização da região como forma de valorizar seus terrenos e, mais tarde, por um processo de especulação, vendê-los a preços mais altos. Desta forma, o centro urbano era rodeado por lotes desocupados que permaneceram nesta situação por décadas. Eventualmente, na década de 1970, o acesso à terra se tornou impossível para a população mais pobre, devido aos altos preços dos lotes em torno do centro urbano. Assim, os mais pobres tiveram que se estabelecer em terrenos mais distantes, dando início ao processo de espraiamento da cidade. Mais tarde, na década de 1980, os lotes intermediários finalmente começaram a ser ocupados, o que retardou ainda mais a possibilidade de investimentos nas regiões

periféricas, já que os lotes intermediários, por estarem mais próximos do centro urbano, começaram a ser atendidos primeiro.

Como o transporte público chegou mais cedo aos terrenos mais próximos do centro urbano, isso os valorizou ainda mais, incentivando o investimento em construções verticais. Evidentemente, isso retarda ainda mais o atendimento às regiões periféricas, pois surge a necessidade de investir mais no centro urbano para atender a uma população cada vez mais densa.

Os autores argumentam que este processo não desencadeou o sucateamento do serviço de trólebus por acaso. Dizem eles que a CTA

> [...] foi criada como uma sociedade anônima, e a composição de capital foi realizada através de uma cobrança adicional no Imposto Predial e Territorial Urbano (IPTU) [...] [11],

o que significa que quem financiava o serviço era a população, incluindo os moradores dos bairros espraiados, aos quais o serviço sequer chegava. Outra consequência importante deste modelo é que os maiores proprietários de terra eram também os maiores acionistas da empresa. Com o desinteresse crescente dos mais pobres em suas ações, a empresa foi ficando cada vez mais concentrada justamente nas mãos daqueles a quem menos interessava expandir o serviço da CTA para as regiões espraiadas. Com o tempo, isso levou ao fortalecimento das empresas privadas de transporte e, eventualmente, a CTA começou a dar prejuízo. Este fato foi utilizado para justificar a privatização da empresa, em 2016, além de sua concessão a um consórcio formado por duas empresas privadas.

Este exemplo mostra que o espraiamento, apesar de ser um fenômeno comum no Brasil, não é inevitável. Apesar de ser interessante modelar e analisar matematicamente este problema, o caso de Araraquara serve para instigar a seguinte pergunta: De que maneira podemos utilizar as ciências sociais para prevenir a consolidação deste tipo de fenômeno no território brasileiro? Afinal, neste caso, seria mais interessante se o fenômeno do espraiamento constituísse uma situação meramente imaginária – se fosse apenas o delírio de um matemático aplicado, em vez de algo que se verifica na realidade. Pelo contrário, o exemplo de Araraquara nos remete ao que escreveu Sérgio Buarque de Holanda [43]:

> A democracia no Brasil foi sempre um lamentável mal-entendido. Uma aristocracia rural e semifeudal importou-a e tratou de acomodá-la, onde fosse possível, aos seus direitos ou privilégios, os mesmos privilégios que tinham sido, no Velho Mundo, o alvo da luta da burguesia contra os aristocratas.

3. Modelagem matemática para roteirização

Problemas de roteirização de ônibus podem ser resolvidos por modelagem, seguida de implementação em ambiente computacional. Para a compreensão dos modelos, é necessário entender conceitos básicos de teoria dos grafos. Para entender como a resolução dos problemas funciona, um aprofundamento matemático em álgebra linear é importante.

O conteúdo de grafos desta seção toma como base os materiais de Feofiloff, Kohayakawa e Wakabayashi [31] e Prestes [60]. Já o conteúdo de álgebra linear é baseado, principalmente, em Bazaraa, Jarvis e Sherali [5], Bertsimas e Tsitsiklis [9] e Ferris, Mangasarian e Wright [32].

3.1. Grafos e dígrafos simples

Definição 3.1 (Grafo simples)**.** Um *grafo simples* $G = (\mathcal{V}, \mathcal{A})$ é uma estrutura discreta formada por um conjunto de vértices $\mathcal{V}$, com $\mathcal{V} \neq \emptyset$, e um conjunto de arestas $\mathcal{A} \subseteq \mathcal{P}(\mathcal{V})$, com $\mathcal{P}(\mathcal{V}) = \{\{x, y\} \mid x \neq y \wedge x, y \in \mathcal{V}\}$. No máximo uma aresta é associada a cada par de vértices.

Devido à natureza dos problemas de roteirização, boa parte dos autores trata os pontos a serem visitados como vértices, com os deslocamentos entre pontos sendo representados por arestas. A Figura 1 ilustra um grafo da forma $G = (\mathcal{V}, \mathcal{A})$.

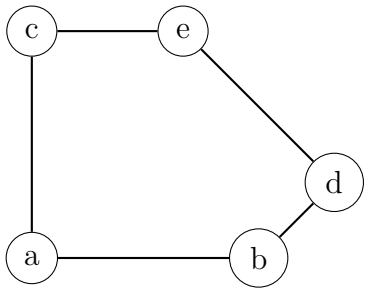


Figura 1

Quando uma aresta conecta dois vértices a e b, diz-se que ela *incide* sobre a e b. A definição de grafo simples não especifica um sentido de conexão entre os pontos. Para situações em que o sentido importa, recorre-se a *dígrafos simples*.

Definição 3.2 (Dígrafo simples)**.** Um *dígrafo simples*, ou *grafo orientado*, $D = (\mathcal{V}_d, \mathcal{A}_d)$ é um grafo com conjunto de vértices $\mathcal{V}_d$ e conjunto de arestas $\mathcal{A}_d \subseteq \mathcal{P}_d(\mathcal{V})$, com $\mathcal{P}_d(\mathcal{V}) = \{(x, y) \mid x \neq y \wedge x, y \in \mathcal{V}_d\}$. Para cada par de vértices x e y, podem existir no máximo

duas arestas relacionando os vértices. Neste caso, uma deve ser (x, y) e a outra deve ser (y, x).

É comum referir-se às arestas de um dígrafo como *arcos*. Os arcos são representados por pares ordenados porque a ordem de deslocamento entre vértices importa. Assim, (x, y) indica deslocamento com origem em x e destino em y, e (y, x) indica deslocamento na direção contrária.

A Figura 2 ilustra um dígrafo simples. Na representação visual, a ponta da flecha do arco indica o destino, e a outra extremidade indica a origem.

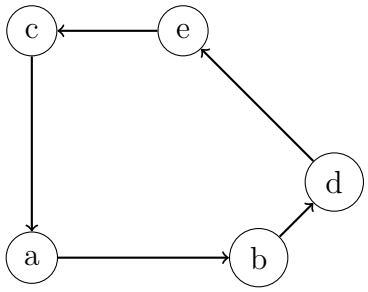
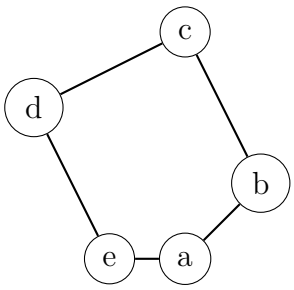


Figura 2

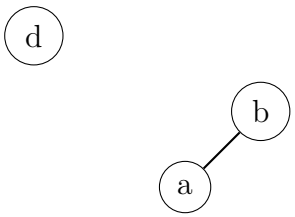
Definição 3.3 (Subgrafo)**.** Um grafo $G_1 = (\mathcal{V}_1, \mathcal{A}_1)$ é um *subgrafo* de $G_2 = (\mathcal{V}_2, \mathcal{A}_2)$, denotado $G_1 \subseteq G_2$, se e só se $\mathcal{V}_1 \subseteq \mathcal{V}_2$ e $\mathcal{A}_1 \subseteq \mathcal{A}_2$. Neste caso, G_1 é *supergrafo* de G_1.

Definição 3.4 (Subgrafo induzido por vértices)**.** Sejam $G = (\mathcal{V}, \mathcal{A})$ um grafo e $S_v \subseteq \mathcal{V}$ um subconjunto não-vazio. Dizemos que um subgrafo de G é *induzido* por S_v (denotado por $G[S_v]$) se seu conjunto de vértices é S_v e seu conjunto de arestas é composto por todas as arestas de G que possuem ambas as extremidades em S_v.

As Figuras 3a e 3b ilustram um grafo e um de seus subgrafos, respectivamente.



(a) Grafo $G = (V, A)$.



(b) Subgrafo induzido $G[\{a, b, d\}]$.

3.2. Conexões entre vértices

Definição 3.5 (Vértices vizinhos)**.** Dado um grafo simples $G = (\mathcal{V}, \mathcal{A})$, dizemos que dois vértices $v_i, v_j \in \mathcal{V}$ são *vizinhos* se existe aresta $\{v_i, v_j\} \in \mathcal{A}$. No caso de um dígrafo simples $D = (\mathcal{V}_d, \mathcal{A}_d)$, dois vértices são vizinhos se existe arco $(v_i, v_j) \in \mathcal{A}_d$ ou $(v_j, v_i) \in \mathcal{A}_d$. Em ambos os casos, estamos assumindo que $v_i \neq v_j$.

A *função de vizinhança*, $\tau : V \to 2^V$, de um grafo simples $G = (\mathcal{V}, \mathcal{A})$, é

$$\tau(v) = \{w \mid \{v, w\} \in \mathcal{A}\}, \tag{1}$$

em que 2^V é o conjunto potência de V.

Já para um dígrafo simples $D = (\mathcal{V}_d, \mathcal{A}_d)$, existem duas funções de vizinhança: a *direta* ($\tau^+ : \mathcal{V} \to 2^{\mathcal{V}_d}$) e a *inversa* ($\tau^- : \mathcal{V} \to 2^{\mathcal{V}_d}$), definidas, respectivamente, a seguir.

$$\tau^+(v) = \{w \mid (v, w) \in \mathcal{A}_d\} \tag{2}$$

e

$$\tau^-(v) = \{w \mid (w, v) \in \mathcal{A}_d\}. \tag{3}$$

A função de vizinhança serve como base para o conceito de *grau*.

Definição 3.6 (Grau de um vértice)**.** Para um grafo simples $G = (\mathcal{V}, \mathcal{A})$, o *grau* de um vértice v, denotado por $d(v)$, é igual ao seu número de vizinhos. Ou seja, $d(v) = |\tau(v)|$.

No caso de um dígrafo simples $D = (\mathcal{V}_d, \mathcal{A}_d)$, o *grau de entrada* $d^-(v)$ e o *grau de saída* $d^+(v)$ são definidos, respectivamente, como $d^-(v) = |\tau^-(v)|$ e $d^+(v) = |\tau^+(v)|$.

Representaremos o maior e o menor grau de todos os vértices em um grafo G, respectivamente, como

$$\Delta(G) = \max_{v \in \mathcal{V}} d(v) \tag{4}$$

e

$$\delta(G) = \min_{v \in \mathcal{V}} d(v). \tag{5}$$

As incidências sobre os vértices de um grafo são passíveis de representação matricial.

Definição 3.7 (Matriz de adjacência)**.** Para um grafo simples $G = (\mathcal{V}, \mathcal{A})$, com $\mathcal{V} = \{v_1, \ldots, v_n\}$, a *matriz de adjacência*, M, é uma matriz $n \times n$ simétrica cujos elementos obedecem à seguinte definição:

$$m_{ij} = \begin{cases} 0, \text{ se } v_i = v_j \text{ ou se } v_i \text{ não é adjacente a } v_j \\ 1, \text{ se } v_i \text{ é adjacente a } v_j. \end{cases} \tag{6}$$

No caso de um dígrafo simples $D = (\mathcal{V}_d, \mathcal{A}_d)$ com $\mathcal{V} = \{v_1, \ldots, v_n\}$, a matriz de adjacência, A, não necessariamente é simétrica, e seus elementos são definidos como:

$$a_{ij} = \begin{cases} 0, \text{ se } v_i = v_j \text{ ou se } v_i \text{ não é origem de um arco que tem destino em } v_j \\ 1, \text{ se } v_i \text{ é origem de um arco que tem destino em } v_j. \end{cases} \tag{7}$$

Definição 3.8 (Ciclo)**.** Em um grafo, um *ciclo* a partir de v é uma sequência alternada de vértices e arestas $v = v_0, a_1, v_1, a_2, v_2, \ldots, a_n, v_n = v$ que começa e termina em v, com todos os vértices e arestas distintos, exceto v_0 e v_n, e com cada aresta a_i incidente aos vértices v_{i-1} e v_i.

Chamamos de *cintura* o menor ciclo em um grafo G, denotado por $g(G)$, e de *circunferência* o seu maior ciclo, denotado por $c(G)$. Um ciclo com n vértices distintos precisa de n arestas conectando os vértices. O número de vértices em um ciclo é conhecido como *comprimento* do ciclo.

Definição 3.9 (Ciclo hamiltoniano)**.** Um *ciclo hamiltoniano*, ou *ciclo de espalhamento*, é um ciclo no qual estão inclusos todos os vértices do grafo.

Definição 3.10 (Grafo e dígrafo hamiltonianos)**.** Um grafo ou dígrafo é hamiltoniano se ele possui um ciclo hamiltoniano.

3.3. O problema do caixeiro viajante

O problema do caixeiro viajante (PCV) é o problema de roteirização de formulação mais simples. A descrição informal feita por Saiyed [63] estabelece que

> O problema do caixeiro viajante envolve um vendedor que precisa passar por um conjunto de cidades tomando o caminho mais curto possível e visitando todas as cidades apenas uma vez, e não menos que uma vez, e retornando ao ponto de partida.[5]

[5] The traveling salesman problem involves a salesman who must make a tour of a number of cities using the shortest path available and visit each city exactly once and only once and return to the original starting point.

Seja $G = (\mathcal{V}, \mathcal{A})$ um grafo, em que $\mathcal{V}$ é o conjunto de vértices a visitar e $\mathcal{A}$ é o conjunto de arestas que incidem sobre os vértices entre os quais ocorre deslocamento. O PCV pode ser interpretado como o problema de determinação de uma sequência de vértices e arestas, com começo e fim num mesmo vértice e grau 2 para todos os vértices, formando um ciclo hamiltoniano [78].

Em geral, o PCV é modelado como um problema cujo objetivo é

$$\min \sum_{i,j \in \mathcal{V}} x_{ij} c_{ij}, \tag{8}$$

em que c_{ij} são coeficientes de custo associados ao deslocamento entre os vértices v_i e v_j, e x_{ij} são variáveis que definem se ocorre deslocamento entre os vértices v_i e v_j. As matrizes $C = [c_{ij}]$ e $X = [x_{ij}]$, que reúnem estes elementos, são chamadas, respectivamente, de matriz de custos e matriz de adjacência.

Definição 3.11 (Matriz simétrica)**.** Uma matriz quadrada $A = [a_{ij}]$ é dita *simétrica* quando $a_{ij} = a_{ji}, \forall i, j$.

Dizemos que o PCV é simétrico ou não dependendo da simetria da matriz de custos. Dantzig, Fulkerson e Johnson [25], por exemplo, consideram que um PCV assimétrico requeira as famílias de restrições

$$\sum_{i \in \mathcal{V}} x_{ij} = 1, \ \forall j \in \mathcal{V}, \tag{9}$$

$$\sum_{j \in \mathcal{V}} x_{ij} = 1, \ \forall i \in \mathcal{V}, \tag{10}$$

em que as Restrições (9) e (10) garantem que a cada ponto se chegue e de cada ponto se saia uma única vez, respectivamente, com $x_{ij} \in \{0, 1\}$, com 1 representando a existência de deslocamento de v_i até v_j e 0 representando a ausência de deslocamento neste sentido. Neste caso, a matriz de adjacência define um dígrafo.

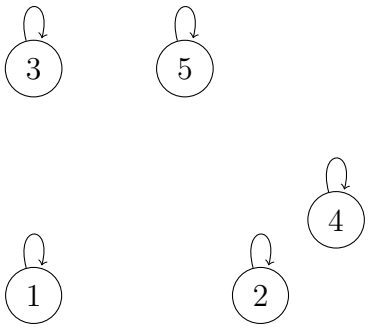
Para o caso simétrico, os autores definem a família de restrições

$$\sum_{j \in \mathcal{V}} x_{ij} = 2, \ \forall i \in \mathcal{V}. \tag{11}$$

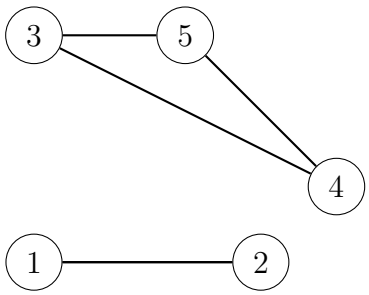
Além de implicar formulações mais simples, o PCV simétrico tem metade do número de soluções possíveis do assimétrico, por não importar o sentido de deslocamento de um

ponto a outro.

As restrições acima não impedem que ocorram soluções em que não é formado ciclo hamiltoniano, como X = I[6], ilustrada na Figura 4a, ou qualquer outra solução que envolva a formação de ciclos de menor comprimento, que chamamos de *subciclos*, como exemplificado na Figura 4b.



(a) Solução com laços.



(b) Solução com subciclos.

Infelizmente, não existem condições necessárias e suficientes[7] para garantir que um grafo tenha ciclo hamiltoniano. No entanto, existem formulações do PCV feitas para impedir a formação de subciclos. Em uma delas, conhecida como DFJ [25], adiciona-se a seguinte família de restrições ao problema:

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \forall S \subsetneq \mathcal{V}. \tag{12}$$

Para completar um ciclo com n vértices, são necessárias n arestas. Assim, esta família de restrições impede a formação de ciclo hamiltoniano em qualquer subgrafo próprio induzido por vértices do problema. Logo, impede-se a formação de subciclos no grafo do problema. No caso da Figura 4a, a formação de laços é impedida por todas as restrições da forma $\sum_{i \in S} x_{ii} \leq 0$, com $S = \{i\}$. No caso da Figura 4b, a formação dos dois subciclos é impedida pelas restrições

$$\sum_{i,j \in S} x_{ij} \leq 1,\ S = \{1, 2\}, \tag{13}$$

[6]Quando $x_{ii} = 1$, $i \in \mathcal{V}$, diz-se que o vértice v_i apresenta um *laço*. Neste caso, a solução encontrada é um *pseudografo*.

[7]Se existissem tais condições, seria possível checá-las para afirmar rapidamente se um grafo tem ciclo hamiltoniano ou não. Se um grafo não satisfizesse as condições necessárias, ele não teria ciclo hamiltoniano; se satisfizesse as condições suficientes, teria. Na ausência destas condições, verificam-se todas as incidências de arestas para fazer esta determinação.

$$\sum_{i,j \in S} x_{ij} \leq 2,\ S = \{3, 4, 5\}. \tag{14}$$

Existem outras formulações populares para o PCV que tentam remover subciclos. Uma delas é a de Miller, Tucker e Zemlin [55], conhecida como MTZ, que adiciona ao problema uma variável u_i para cada vértice do PCV. Estas variáveis têm o propósito de ordenar os pontos visitados, respeitando as famílias de restrições

$$u_i - u_j + n x_{ij} \leq n - 1,\ \forall i \in \mathcal{V},\ j \in \mathcal{V} \backslash \{0\},\ i \neq j, \tag{15}$$

$$u_i \in \mathbb{Z}. \tag{16}$$

Se não existir deslocamento de um ponto i a um ponto j, então $x_{ij} = 0$ e, consequentemente, sobra que $u_i - u_j \leq n - 1$, o que garante que o valor de cada variável de ordenação não exceda o número de pontos permutáveis do problema. Se existir deslocamento de i a j, segue-se que $x_{ij} = 1$ e, desta maneira, $u_i - u_j \leq -1$. Ou seja, o próximo ponto na ordem de deslocamentos deve ficar uma unidade à frente do anterior.

Outra opção é a formulação de Gavish e Graves [36], conhecida como GG, que introduz variáveis de ordenação z_{ij} ao problema, relacionando cada vértice v_i a cada vértice v_j. Esta formulação impõe as seguintes famílias de restrições ao problema:

$$\sum_{j=0} z_{ij} - \sum_{j \neq 0} z_{ji} = 1, \qquad i = 1, \ldots, n, \tag{17}$$

$$z_{ij} \leq (n-1) x_{ij}, \qquad i = 1, \ldots, n,\ j = 0, \ldots, n, \tag{18}$$

$$z_{ij} \geq 0, \qquad \forall i, j. \tag{19}$$

Estas formulações têm diferentes performances quando implementadas em ambiente computacional. Uma discussão detalhada sobre isso é feita na Seção 4

Exemplo 3.12 (Complexidade computacional do PCV). Encontrar a solução ótima de um PCV é algo difícil ou impossível, como descreve Zambito [78]. O PCV se enquadra na categoria de problema NP-difícil, o que significa que só é plausível encontrar soluções exatas até certo número de pontos. Para resolver problemas muito grandes, são empregadas técnicas para encontrar soluções próximas da ótima.

A título de ilustração da complexidade do PCV – e, por extensão, de problemas mais elaborados –, suponha-se o objetivo de determinar a melhor rota de um PCV calculando

todas as rotas possíveis e selecionando a que apresentar menor distância total – ideia que será denominada *método de força bruta*.

Quantas rotas existem em um PCV assimétrico? Os n pontos além do depósito são permutáveis, portanto, existem $n!$ ciclos hamiltonianos possíveis. A distância total de cada ciclo é determinada por n somas de custos de vértice a vértice, o que requer, no mínimo, um total de $n!n$ operações computacionais para que se possa determinar o melhor[8]. Supondo tempo de soma individual ε, tem-se os tempos de computação da Tabela 1.

Tabela 1: Fator de aumento do tempo mínimo de computação de um número de pontos para outro no PCV.

Pontos	Tempo (ε)	Fator de aumento
5	96	–
6	600	6.25
7	4320	7.2
8	35280	8.17
9	322560	9.14
10	3265920	10.13

Nestas condições, um PCV de 10 pontos leva no mínimo 34 mil vezes mais tempo para resolver do que um de 5 pontos. Para um exemplo de implementação e execução do método de força bruta, ver [69].

3.4. Problemas de otimização

O modelo de PCV simétrico da Seção 3.3 pode ser escrito como

$$\min \sum_{i,j\in\mathcal{V}} x_{ij}c_{ij}, \tag{20}$$

$$\text{s.a} \sum_{j\in\mathcal{V}} x_{ij} = 2, \qquad \forall i \in \mathcal{V}, \tag{21}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1, \qquad \forall S \subsetneq \mathcal{V}, \tag{22}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in \mathcal{V}. \tag{23}$$

[8] A alocação de memória necessária é um fator que não é contabilizado neste exemplo.

A Expressão (20) define o PCV como um problema de *minimização*, o que o coloca na categoria de problema de *otimização*[9]. Considerando-se as Restrições (21) e (22), tem-se um problema de otimização *linear*. Este tipo de problema aparece na *forma padrão*

$$\min z \coloneqq c^{\intercal}x \tag{24}$$
$$\text{s.a } Ax \leq b, \tag{25}$$
$$x \geq \mathbf{0}, \tag{26}$$

em que $c \in \mathbb{R}^n$ é o vetor de coeficientes de custo, $x \in \mathbb{R}^n$ é o vetor de variáveis de decisão, $A \in \mathbb{R}^{m \times n}$ é a matriz de coeficientes de restrições e $b \in \mathbb{R}^m$ é o vetor do lado direito das inequações do problema. Chama-se z de *valor objetivo* ou *função objetivo*. Em geral, é desejável ter $b \geq \mathbf{0}$. Assim, para todo elemento $b_i \leq 0$, multiplica-se a inequação $A_i x \leq b_i$ por -1, invertendo o sinal da inequação.

A *forma canônica* de um problema de otimização linear é

$$\min z \coloneqq c^{\intercal}x \tag{OL}$$
$$\text{s.a } Ax = b, \tag{27}$$
$$x \geq \mathbf{0}. \tag{28}$$

Para converter um problema para a forma canônica, adicionam-se ou subtraem-se variáveis não-negativas de todas as inequações, transformando-as em equações. Tais variáveis são chamadas de *variáveis de folga*.

- Se $A_i x \leq b$, então adiciona-se $x_s \geq 0$ tal que $A_i x + x_s = b_i$;
- Se $A_i x \geq b$, então subtrai-se $x_s \geq 0$ tal que $A_i x - x_s = b_i$.

A conversão para a forma canônica é feita para possibilitar sua resolução pelo método simplex. Este método foi desenvolvido pelo matemático George Dantzig na década de 40. Os avanços em computação financiados pelo departamento de defesa dos Estados Unidos na época incentivaram as pesquisas de Dantzig e outros matemáticos [27]. As pesquisas em otimização linear renderam frutos em várias indústrias e processos logísticos, sendo cruciais no campo da pesquisa operacional [70].

[9] Outra palavra muito utilizada para se referir a otimização é *programação*. Preferimos evitar o uso deste termo devido à possibilidade de confusão com programação de computadores.

Apesar da sua grande utilidade, o método simplex se limita a resolver problemas de otimização linear. Como dito no começo desta seção, o PCV só é linear quando as Restrições (23) são desconsideradas. Este tipo de técnica, em que restrições que forçam variáveis a serem inteiras são descartadas, é chamado de *relaxação* do problema [7].

Quando o modelo completo do PCV é considerado, tem-se um problema de otimização *inteira*, dado que todas as variáveis de decisão são inteiras. Para problemas mais complexos, é possível fazer uma mistura de variáveis inteiras com variáveis reais, o que categoriza um problema de otimização *inteira mista* [13, 46, 49].

A álgebra linear desenvolvida no restante desta seção é necessária para compreender os alicerces do método simplex e as implementações computacionais realizadas neste trabalho. Para uma explicação sobre os procedimentos do simplex, consultar o Apêndice A.

3.5. Poliedros e convexidade

Definição 3.13 (Poliedro)**.** Um *poliedro* $\mathbf{P}$ é um conjunto que pode ser escrito na forma $\mathbf{P} \coloneqq \{x \in \mathbb{R}^n \mid Ax \leq b\}$, onde A é uma matriz $m \times n$ e b é um vetor em $\mathbb{R}^m$.

O conjunto de restrições que definem um problema de otimização linear (OL) pode ser interpretado como um poliedro. Esta interpretação geométrica facilita a compreensão dos problemas. É possível interpretar as próprias restrições como objetos geométricos, sendo o poliedro o resultado da intersecção destes objetos. Assim, todo ponto que satisfizer todas as restrições do poliedro é dito *factível* [51].

Definição 3.14 (Hiperplano)**.** Seja a um vetor não-nulo em $\mathbb{R}^n$ e seja b um escalar. Então o conjunto $\mathbf{H} \coloneqq \{x \in \mathbb{R}^n \mid a^\intercal x = b\}$ é chamado de *hiperplano*.

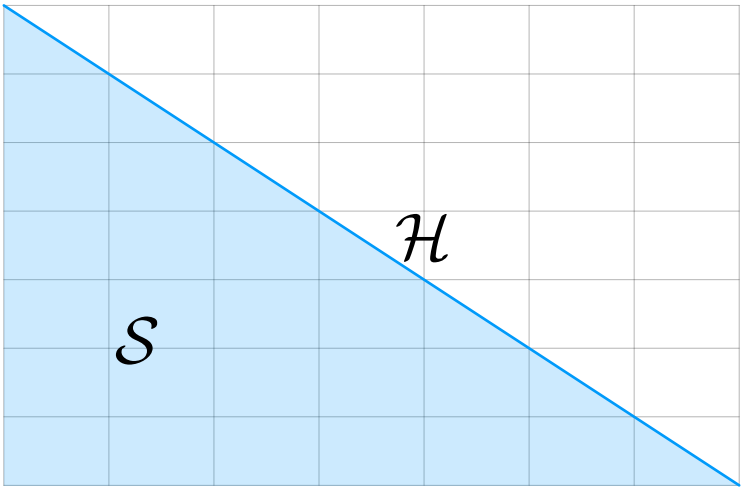
Definição 3.15 (Semiespaço)**.** Seja a um vetor não-nulo em $\mathbb{R}^n$ e seja b um escalar. Então o conjunto $\mathbf{S} \coloneqq \{x \in \mathbb{R}^n \mid a^\intercal x \leq b\}$ é chamado de *semiespaço*.

Hiperplanos e semiespaços são generalizações dos conceitos tridimensionais de aresta e espaço. Um exemplo visual destes conceitos é apresentado na Figura 5. O plano em azul transparente é um semiespaço definido por uma desigualdade $a^\intercal x \leq b$, enquanto o hiperplano é a reta que delimita este semiespaço na igualdade $a^\intercal x = b$.

A convexidade é uma propriedade fundamental para poliedros passíveis de resolução pelo método simplex. A seguir, são apresentados alguns conceitos e teoremas a respeito desta propriedade.

Definição 3.16 (Conjunto convexo)**.** Um conjunto $\mathbf{S} \subset \mathbb{R}^n$ é *convexo* se, para todo $x, y \in \mathbf{S}$, e qualquer $\lambda \in [0, 1]$, tem-se $\lambda x + (1 - \lambda)y \in \mathbf{S}$.

Figura 5: Exemplo de hiperplano ($\mathcal{H}$) e semiespaço ($\mathcal{S}$)



Definição 3.17 (Combinação convexa e envoltória convexa)**.** Sejam $x^1, \ldots, x^k$ vetores em $\mathbb{R}^n$ e sejam $\lambda_1, \ldots, \lambda_n$ escalares não-negativos cuja soma é um. Então:

(a) O vetor $\sum_{i=1}^{k} \lambda_i x^i$ é uma *combinação convexa* dos vetores $x^1, \ldots, x^k$.

(b) A *envoltória convexa* dos vetores $x^1, \ldots, x^k$ é o conjunto de todas as combinações convexas desses vetores.

Teorema 3.18. *(a) A intersecção de conjuntos convexos é convexa.*

(b) Todo poliedro é um conjunto convexo.

(c) Uma combinação convexa de um número finito de elementos de um conjunto convexo pertence ao conjunto convexo.

(d) A envoltória convexa de um número finito de vetores é um conjunto convexo.

Demonstração. (a) Sejam $\mathbf{S_i}, i \in I$ conjuntos convexos onde I é um conjunto de índices, e suponha que x e y pertencem à intersecção $\cap_{i \in I} \mathbf{S_i}$. Seja $\lambda \in [0, 1]$. Como cada $\mathbf{S_i}$ é convexo e contém x e y, temos $\lambda x + (1 - \lambda)y \in \mathbf{S_i}$, o que prova que $\lambda x + (1 - \lambda)y$ também pertence à intersecção dos conjuntos $\mathbf{S_i}$. Logo, $\cap_{i \in I} \mathbf{S_i}$ é convexo.

(b) Sejam a um vetor e b um escalar. Suponha que x e y satisfazem $a^\intercal x \leq b$ e $a^\intercal y \leq b$, respectivamente, e logo pertencem ao mesmo semiespaço. Seja $\lambda \in [0, 1]$. Então $a^\intercal(\lambda x + (1 - \lambda)y) \leq \lambda b + (1 - \lambda)b = b$, o que prova que $\lambda x + (1 - \lambda)y$ também pertence ao mesmo semiespaço. Logo, um semiespaço é convexo. Como um poliedro

é uma intersecção de um número finito de semiespaços, o resultado é confirmado pelo Item (a).

(c) Uma combinação convexa de dois elementos de um conjunto convexo está contida no conjunto, pela definição de convexidade. Suponhamos, por hipótese indutiva, que uma combinação convexa de k elementos de um conjunto convexo pertence ao conjunto. Considere $k+1$ elementos $x^1, \ldots, x^{k+1}$ de um conjunto convexo $\mathbf{S}$ e sejam $\lambda_1, \ldots, \lambda_{k+1}$ escalares não-negativos de soma um. Assumimos, sem perda de generalidade, que $\lambda_{k+1} \neq 1$. Então temos:

$$\sum_{i=1}^{k+1} \lambda_i x^i = \lambda_{k+1} x^{k+1} + (1 - \lambda_{k+1}) \sum_{i=1}^{k} \frac{\lambda_i}{1 - \lambda_{k+1}} x^i. \tag{29}$$

Note que $\sum_{i=1}^{k} \frac{\lambda_i}{1-\lambda_{k+1}} = 1$, pois, como definido anteriormente, $\sum_{i=1}^{k+1} \lambda_i = 1$, o que implica que $(\sum_{i=1}^{k+1} \lambda_i) - \lambda_{k+1} = 1 - \lambda_{k+1} = \sum_{i=1}^{k} \lambda_i$. Usando a hipótese indutiva, $\sum_{i=1}^{k} \frac{\lambda_i x^i}{1-\lambda_{k+1}} \in \mathbf{S}$. Então, o fato de $\mathbf{S}$ ser convexo e a Equação (29) implicam que $\sum_{i=1}^{k+1} \lambda_i x^i \in \mathbf{S}$, e a indução está completa.

(d) Seja $\mathbf{S}$ a envoltória convexa dos vetores $x^1, \ldots, x^k$, e sejam $y = \sum_{i=1}^{k} \xi_i x^i, z = \sum_{i=1}^{k} \theta_i x^i$ dois elementos de $\mathbf{S}$, onde $\xi_i \geq 0, \theta_i \geq 0$ e $\sum_{i=1}^{k} \xi_i = \sum_{i=1}^{k} \theta_i = 1$. Seja $\lambda \in [0, 1]$. Então, $\lambda y + (1-\lambda) z = \lambda \sum_{i=1}^{k} \xi_i x^i + (1-\lambda) \sum_{i=1}^{k} \theta_i x^i = \sum_{i=1}^{k} (\lambda \xi_i + (1-\lambda)\theta_i) x^i$.

Note que os coeficientes $\lambda \xi_i + (1-\lambda)\theta_i, i = 1, \ldots, k$ são não-negativos e somam um. Isso mostra que $\lambda y + (1-\lambda) z$ é uma combinação convexa de $x^1, \ldots, x^k$ e, portanto, pertence a $\mathbf{S}$. Isso estabelece a convexidade de $\mathbf{S}$.

□

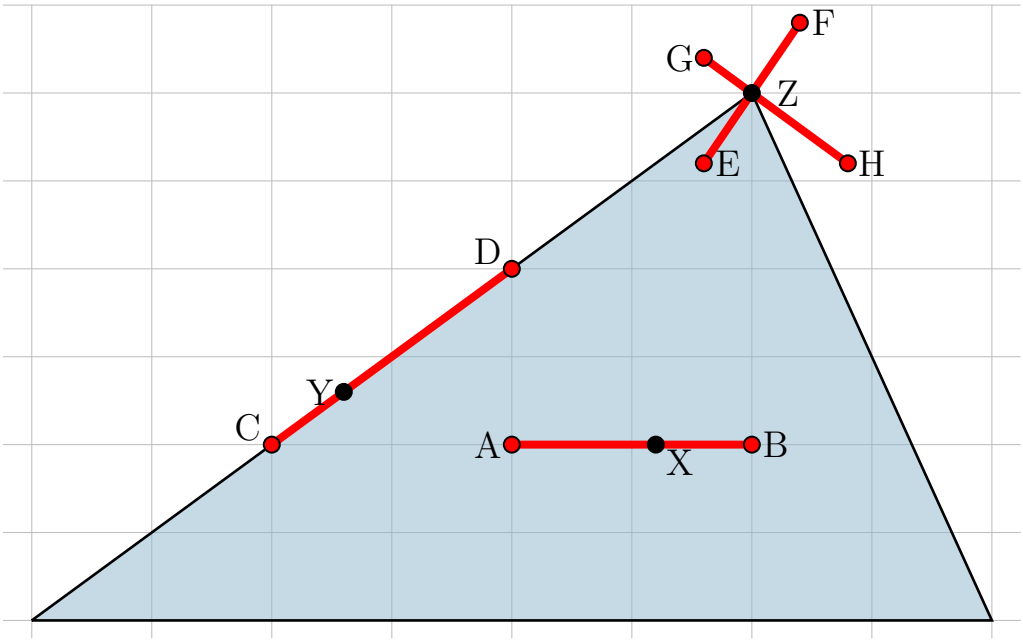
3.6. Ponto extremo e vértice

Os conceitos de ponto extremo e vértice são fundamentais em otimização linear. Como é provado na Seção 3.8, estes conceitos são equivalentes entre si, além de serem equivalentes ao conceito de solução básica factível, introduzido na Seção 3.7. Como é provado no Apêndice A.1, a solução ótima de um problema com restrições lineares se encontra sempre em um de seus vértices, quando o poliedro do problema é limitado.

Definição 3.19 (Ponto extremo)**.** Seja $\mathbf{P}$ um poliedro. Um vetor $x \in \mathbf{P}$ é um *ponto extremo* de $\mathbf{P}$ se não existem dois vetores $y, z \in \mathbf{P}$, ambos diferentes de x, e um escalar $\lambda \in [0, 1]$, tais que $x = \lambda y + (1 - \lambda) z$.

Uma interpretação possível de pontos extremos pode ser obtida desenhando a ponta de um poliedro e seus arredores.

Figura 6: Exemplo visual de ponto extremo



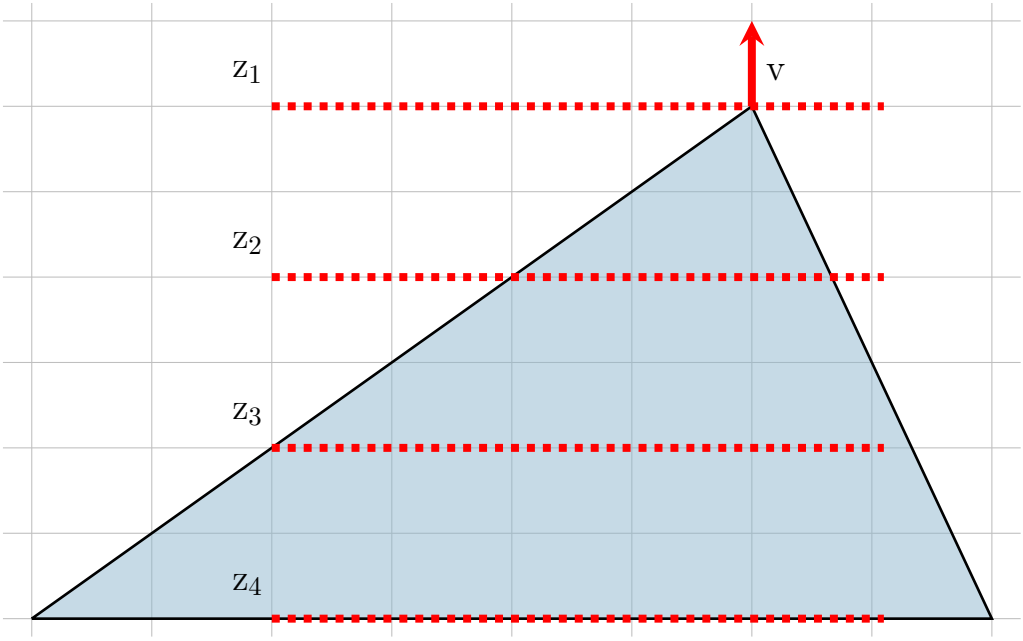
O exemplo da Figura 6 possui um poliedro, em azul, três pontos pretos que se espera representar por combinação convexa e quatro combinações de pontos, em vermelho. O ponto X pode ser representado por combinação convexa dos pontos A e B, pois ambos se encontram dentro do poliedro. O ponto Y, localizado numa das arestas do poliedro, também pode ser representado por combinação convexa de dois pontos, neste caso C e D, ambos localizados na mesma aresta que Y.

O que não podemos representar por combinação convexa de dois pontos é o ponto Z, que se localiza justamente numa das pontas do poliedro. As únicas combinações cujos segmentos de reta resultantes passam por Z contêm um ponto dentro e outro fora do poliedro (E e F, por exemplo) ou dois pontos fora do poliedro (G e H, por exemplo). Portanto, dos três pontos pretos, apenas Z não pode ser representado como combinação convexa, e, logo, ele é um ponto extremo.

Definição 3.20 (Vértice)**.** Seja $\mathbf{P}$ um poliedro. Um vetor $x \in \mathbf{P}$ é um *vértice* de $\mathbf{P}$ se existir vetor c tal que $c^\intercal x < c^\intercal y$ para qualquer y que satisfaça $y \in \mathbf{P}$ e $y \neq x$.

Geometricamente, os vértices podem ser percebidos conforme a Figura 7.

Figura 7: Exemplo visual de vértice



O vetor v define infinitos hiperplanos cujos valores crescem no sentido do vetor. O conjunto desses hiperplanos define um semiespaço. Desta maneira, temos que $z_1 > z_2 > z_3 > z_4$. Perceba que os valores z_2, z_3 e z_4 podem ser obtidos multiplicando v por infinitos pontos do poliedro, justamente os que estão nas linhas tracejadas que representam os respectivos hiperplanos. Escolhemos um de cada, que denotaremos pelos vetores x^2, x^3 e x^4. Ou seja, $v^\intercal x^2 = z_2$, $v^\intercal x^3 = z_3$ e $v^\intercal x^4 = z_4$. No entanto, existe um único vetor do poliedro através do qual se pode obter z_1, que é o vetor que descreve a sua ponta, x^1. Portanto, existe um único ponto do poliedro cujo produto interno com o vetor v é maior do que seu produto interno com qualquer outro ponto do poliedro, e, então, este ponto só pode ser um vértice.

3.7. Solução básica factível

Definição 3.21 (Espaço vetorial)**.** Um *espaço vetorial* é um conjunto cujos elementos, chamados vetores, podem ser somados e multiplicados por números escalares.

Definição 3.22 (Subespaço)**.** Um subconjunto não-vazio $\mathbf{S}$ de $\mathbb{R}^n$ é chamado de *subespaço* de $\mathbb{R}^n$ se $ax + by \in S$ para todo $x, y \in \mathbf{S}$ e todo $a, b \in \mathbb{R}$. Se $\mathbf{S} \neq \mathbb{R}^n$, $\mathbf{S}$ é um *subespaço próprio*. Todo subespaço precisa conter o vetor nulo.

Definição 3.23 (Espaço gerado)**.** O *espaço gerado (span)* de um número finito de vetores

$x^1, \ldots, x^K$ em $\mathbb{R}^n$ é um subespaço de $\mathbb{R}^n$ definido como o conjunto de todos os vetores y da forma $y = \sum_{k=1}^{K} a_k x^k$, onde cada a_k é um número real. Qualquer vetor y assim é chamado de *combinação linear* de $x^1, \ldots, x^K$.

Definição 3.24 (Base)**.** Dado um subespaço $\mathbf{S}$ de $\mathbb{R}^n$, com $\mathbf{S} \neq \{0\}$, uma *base* de $\mathbf{S}$ é uma coleção de vetores que são linearmente independentes e cujo espaço gerado é igual a $\mathbf{S}$. Toda base de um subespaço tem o mesmo número de vetores, que é chamado de *dimensão* do subespaço. Em particular, a dimensão de $\mathbb{R}^n$ é n, e todo subespaço próprio de $\mathbb{R}^n$ tem uma dimensão menor do que n.

Definição 3.25 (Subespaço afim)**.** Seja $\mathbf{S_0}$ um subespaço de $\mathbb{R}^n$ e seja x^0 algum vetor. O vetor $\mathbf{S}$ definido como

$$\mathbf{S} = x^0 + \mathbf{S_0} = \{x + x^0 \mid x \in \mathbf{S_0}\}$$

é chamado de *subespaço afim*, e corresponde a uma translação do subespaço $\mathbf{S_0}$. Por definição, a dimensão de $\mathbf{S}$ é igual à dimensão de $\mathbf{S_0}$.

Teorema 3.26. *Suponha que o espaço gerado* $\mathbf{S}$ *dos vetores* $x^1, \ldots, x^K$ *tem dimensão* m*. Então:*

(a) Existe uma base de $\mathbf{S}$ *consistindo de* m *dos vetores* $x^1, \ldots, x^K$*.*

(b) Se $k \leq m$ *e* $x^1, \ldots, x^k$ *são linearmente independentes, podemos formar uma base de* $\mathbf{S}$ *começando com* $x^1, \ldots, x^k$ *e escolhendo* $m - k$ *dos vetores* $x^{k+1}, \ldots, x^K$*.*

Demonstração. Vamos começar provando o teorema pelo item (b). A princípio, se todos os vetores $x^{k+1}, \ldots, x^K$ puderem ser representados como combinações lineares dos vetores $x^1, \ldots, x^k$, então os vetores $x^1, \ldots, x^k$ formam uma base do subespaço e, portanto, $k = m$. Do contrário, podemos encontrar um vetor em $x^{k+1}, \ldots, x^K$ que seja linearmente independente de todos os vetores em $x^1, \ldots, x^k$, aumentando para k + 1 o número de vetores linearmente independentes. Repetindo este processo $m - k$ vezes, obtemos a base desejada de $\mathbf{S}$. O item (a) decorre trivialmente do caso em que começamos com $k = 0$ no processo que acabamos de apresentar. □

Com estas definições, podemos retornar aos pontos extremos e aos vértices. O fato é que as definições desses dois conceitos são de difícil implementação computacional. No caso de pontos extremos, precisaríamos encontrar uma forma de garantir ao computador que nenhuma de uma infinidade de possíveis combinações de dois pontos é igual ao ponto

que se deseja provar extremo. Já no caso de vértices, há uma infinidade de vetores passíveis de teste.

Para contornar este problema, introduziremos a definição algébrica de soluções básicas factíveis. Para tal, primeiro vamos considerar um poliedro $\mathbf{P} \subset \mathbb{R}^n$ definido em termos de restrições de igualdade e desigualdade:

$$\begin{aligned} a_i^\intercal x &\geq b_i, \quad i \in \mathbf{M_1}, \\ a_i^\intercal x &\leq b_i, \quad i \in \mathbf{M_2}, \\ a_i^\intercal x &= b_i, \quad i \in \mathbf{M_3}. \end{aligned}$$

Em que $\mathbf{M_1}, \mathbf{M_2}$ e $\mathbf{M_3}$ são conjuntos finitos de índices, cada a_i é um vetor em $\mathbb{R}^n$, e cada b_i é um escalar.

Definição 3.27 (Restrição ativa)**.** Se um vetor x^* satisfaz $a_i^\intercal x^* = b_i$ para algum i em $\mathbf{M_1}, \mathbf{M_2}$ ou $\mathbf{M_3}$, dizemos que a restrição correspondente é *ativa* (*active* ou *binding*) em x^*.

Se existirem n restrições que são ativas em um vetor x^*, então x^* satisfaz um certo sistema de n equações lineares com n incógnitas. Este sistema só terá solução única se as n equações forem linearmente independentes. Disso, segue o teorema:

Teorema 3.28. *Seja x^* um elemento de $\mathbb{R}^n$ e seja $I(x^*) = \{i \mid a_i^\intercal x = b_i\}$ o conjunto dos índices das restrições que são ativas em x^*. Então as seguintes afirmações são equivalentes:*

(a) Existem n vetores no conjunto $\{a_i \mid i \in I\}$, que são linearmente independentes.

(b) O espaço gerado dos vetores $a_i, i \in I$ é todo o $\mathbb{R}^n$; ou seja, todo elemento de $\mathbb{R}^n$ pode ser expresso como uma combinação linear dos vetores $a_i, i \in I$.

(c) O sistema de equações $a_i^\intercal x = b_i, i \in I$ tem uma única solução.

Demonstração. Suponha que os vetores $a_i, i \in I$ gerem o espaço $\mathbb{R}^n$. Então, o espaço gerado desses vetores tem dimensão n. Pelo Teorema 3.26, n desses vetores formam uma base de $\mathbb{R}^n$ e são, portanto, linearmente independentes. Por outro lado, suponha que n dos vetores $a_i, i \in I$ são linearmente independentes. Então, o subespaço gerado por esses n vetores é n-dimensional e precisa ser igual a $\mathbb{R}^n$. Logo, todo elemento de $\mathbb{R}^n$ é uma combinação linear dos vetores $a_i, i \in I$. Isso estabelece a equivalência do Item (a) e do Item (b).

Se o sistema de equações tem múltiplas soluções – por exemplo, x^1 e x^2 – então o vetor não-nulo $d = x^1 - x^2$ satisfaz $a_i^\intercal d = 0$ para todo $i \in I$. Como d é ortogonal a todo $a_i, i \in I$, d não é uma combinação linear desses vetores e logo os vetores $a_i, i \in I$

não geram $\mathbb{R}^n$. Reciprocamente, se os vetores $a_i, i \in I$ não geram $\mathbb{R}^n$, escolhemos um vetor não-nulo d que seja ortogonal ao subespaço gerado por esses vetores. Se x satisfaz $a_i^\intercal x = b_i$ para todo $i \in I$, também temos $a_i^\intercal (x + d) = b_i$ para todo $i \in I$, obtendo, assim, várias soluções. Logo, estabelecemos a equivalência entre o Item (b) e o Item (c). □

A partir daqui, faremos um abuso de linguagem e diremos que uma restrição será linearmente independente quando seu vetor a_i for linearmente independente dos vetores a_i das demais restrições.

Definição 3.29 (Solução básica e solução básica factível)**.** Considere um poliedro **P** definido por restrições lineares de igualdade e desigualdade, e seja x^* um elemento de $\mathbb{R}^n$.

- O vetor x^* é uma *solução básica* se todas as restrições de igualdade são ativas nele e se, de todas as restrições ativas em x^*, existem n que são linearmente independentes.
- Se x^* é uma solução básica que satisfaz todas as restrições do poliedro, ela é uma *solução básica factível*.

Como explicado no Apêndice A, o método simplex "salta" de uma solução básica factível a outra de maneira condizente com a definição a seguir.

Definição 3.30 (Soluções básicas adjacentes)**.** Duas soluções básicas distintas no $\mathbb{R}^n$ são *adjacentes* quando existem $n - 1$ restrições linearmente independentes que são ativas em ambas.

Além disso, outra definição que será importante é a de *degeneração*.

Definição 3.31 (Degeneração)**.** Uma solução básica $x \in \mathbb{R}^n$ é *degenerada* se mais do que n restrições são ativas em x.

Por fim, uma última observação importante a ser feita sobre soluções básicas factíveis é o teorema a seguir.

Teorema 3.32. *Para um número finito de restrições lineares de desigualdade, só pode existir um número finito de soluções básicas.*

Demonstração. Considere um sistema de m restrições lineares de desigualdade impostas a um vetor $x \in \mathbb{R}^n$. Em qualquer solução básica, existem n restrições linearmente independentes ativas. Pelo Teorema 3.28, estas n restrições linearmente independentes definem um ponto único. Logo, diferentes soluções básicas correspondem a diferentes combinações de n restrições linearmente independentes ativas e, portanto, correspondem a diferentes pontos únicos. Assim sendo, o número de soluções básicas é limitado pela quantidade de combinações possíveis de n restrições linearmente independentes de um total de m. □

Assim, como diz Santos [64], o número máximo possível de soluções básicas factíveis para um problema é a permutação

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \geq \left(\frac{n}{m}\right)^m, \tag{30}$$

o que evidencia que o número de soluções básicas e básicas factíveis de um problema cresce rapidamente conforme são adicionadas restrições e variáveis.

3.8. Equivalências

Teorema 3.33. *Seja* $\mathbf{P}$ *um poliedro não-vazio e seja* $x^* \in \mathbf{P}$*. Então, as seguintes afirmações são equivalentes:*

(i) x^* *é um vértice;*

(ii) x^* *é um ponto extremo;*

(iii) x^* *é uma solução básica factível.*

Demonstração. Para os propósitos desta prova, e sem perda de generalidade, assumiremos que $\mathbf{P}$ é representado em termos de restrições da forma $a_i^\intercal x \leq b_i$ e $a_i^\intercal x = b_i$.

Vértice $\Rightarrow$ Ponto Extremo

Suponha que $x^* \in \mathbf{P}$ é um vértice. Pela Definição 3.20, existe algum $c \in \mathbb{R}^n$ tal que $c^\intercal x^* < c^\intercal y$ para todo $y \in \mathbf{P}$ tal que $y \neq x^*$. Se $w \in \mathbf{P}, z \in \mathbf{P}, w \neq x^*, z \neq x^*$ e $0 \leq \lambda \leq 1$, então, $c^\intercal x^* < c^\intercal w, c^\intercal x^* < c^\intercal z$, o que implica que $c^T x^* < c^\intercal(\lambda w + (1-\lambda)z)$ e, portanto, $x^* \neq \lambda w + (1-\lambda)z$. Logo, x^* não pode ser representado como combinação convexa de dois outros elementos de $\mathbf{P}$ e, portanto, é um ponto extremo.

Ponto Extremo $\Rightarrow$ Solução Básica Factível

Suponha que $x^* \in \mathbf{P}$ não é uma solução básica factível. Mostraremos que x^* não é ponto extremo de $\mathbf{P}$. Seja $I = \{i \mid a_i^\intercal x = b_i\}$. Como x^* não é solução básica factível, não existem n vetores linearmente independentes na família $a_i, i \in I$, pois, se existissem, x^* teria que estar fora do poliedro e, assim, não faria sentido considerá-lo. Logo, os vetores $a_i, i \in I$ estão em um subespaço do $\mathbb{R}^n$, e existe um vetor não-nulo $d \in \mathbb{R}^n$ tal que $a_i^\intercal d = 0$, para todo $i \in I$.

Seja ϵ um número positivo pequeno e considere os vetores $y = x^* + \epsilon d$ e $z = x^* - \epsilon d$. Perceba que $a_i^\intercal y = a_i^\intercal z = a_i^\intercal x^* = b_i$, para $i \in I$. Ademais, para $i \notin I$, temos $a_i^\intercal x^* < b_i$ e, dado que ϵ é pequeno, também temos $a_i^\intercal y < b_i$. Logo, quando ϵ é suficientemente

pequeno, $y \in P$ e, por argumento similar, $z \in P$. Por fim, note que $x^* = \frac{y+z}{2}$, o que implica que x^* não é ponto extremo.

Solução Básica Factível ⇒ Vértice

Seja x^* uma solução básica factível e seja $I = \{i \mid a_i^\intercal x^* = b_i\}$. Seja $c = \sum_{i \in I} a_i$. Então temos

$$c^\intercal x^* = \sum_{i \in I} a_i^\intercal x^* = \sum_{i \in I} b_i. \tag{31}$$

Ademais, para todo $x \in P$ e todo i, temos $a_i^\intercal x \leq b_i$, e

$$c^\intercal x = \sum_{i \in I} a_i^\intercal x \leq \sum_{i \in I} b_i. \tag{32}$$

Isso mostra que x^* é uma solução ótima para o problema de minimizar $c^\intercal x$ no conjunto $\mathbf{P}$. Ademais, a igualdade é possível na Equação (32) se e somente se $a_i^\intercal x = b_i$, para todo $i \in I$. Como x^* é uma solução básica factível, existem n restrições linearmente independentes que são ativas em x^*, e x^* é solução única para o sistema de equações $a_i^\intercal x = b_i, i \in I$ (Teorema 3.28). Logo, x^* é o minimizador único de $c^\intercal x$ no conjunto $\mathbf{P}$ e, portanto, é um vértice de $\mathbf{P}$. □

3.9. Resumo do método simplex

Nesta seção, é apresentado um resumo do Apêndice A.

Todo problema de otimização linear em cujo poliedro exista pelo menos um ponto possui solução. Quando o poliedro é limitado (isto é, quando não existe semirreta contida no poliedro), a solução ótima para o problema está, necessariamente, localizada em algum dos seus vértices. Nos casos em que o poliedro é ilimitado, além de poder estar em um vértice, a solução ótima pode ser $-\infty$ ou ∞ (para problemas de minimização e maximização, respectivamente). Nestes dois últimos casos, não existe ponto específico que otimize a função objetivo.

O método simplex funciona baseado nestes conhecimentos. Assumindo que a matriz de restrições de um problema tenha posto completo, pode-se resumir o método da seguinte maneira:

(a) Encontra-se um vértice do poliedro do problema, se existir;

(b) Analisam-se os vértices adjacentes ao atual:

1. Se não existir vértice que reduza o valor objetivo, pular para (c);

2. Do contrário, mover-se para o vértice que reduza ao máximo o valor objetivo e repetir (b).

(c) Verificam-se as variáveis de decisão do problema:

1. Se existir variável que possa ser aumentada infinitamente, sem violar a factibilidade do poliedro, a solução ótima é ∞ ou $-\infty$;
2. Do contrário, a solução ótima se encontra no vértice atual.

Um empecilho ao Item (a) é a dificuldade de obtenção de um vértice inicial. A Equação (30) deixa claro que o número de soluções básicas cresce rapidamente com a complexidade do problema tratado, de modo que determinar um vértice inicial por tentativa e erro possa ser ineficiente. O método para encontrar vértices iniciais implementado neste trabalho é o *método das duas fases*, que resolve um novo problema, cujo vértice inicial é conhecido, e cuja solução ótima é um vértice do problema original. Quando um vértice inicial não é encontrado, o problema não tem solução factível.

A degeneração é um problema que afeta o Item (b). Ao encontrar um vértice degenerado, é possível que o simplex entre em um laço infinito em que permaneça no mesmo vértice. Assim como no caso anterior, existem várias formas de resolver este problema. Para propósitos acadêmicos, a *regra lexicográfica* implementada neste trabalho é satisfatória.

Apesar de o método simplex resolver apenas problemas de otimização linear, existem técnicas que permitem que problemas de otimização inteira e inteira mista sejam resolvidos com a ajuda do simplex.

4. Considerações computacionais para roteirização

Nesta seção, discutem-se questões computacionais de implementação e resolução de problemas de otimização linear e inteira.

4.1. Metodologia de modelagem em ambiente computacional

Existem centenas de linguagens de programação, como compilado por Schade, Scheithauer e Scheler [65]. Excluindo linguagens puramente lúdicas, existe ainda uma vasta gama de linguagens com aplicações práticas que podem ser consideradas para uso neste trabalho, dentre as quais as mais populares são reportadas por Carbonnelle [16], baseado no número de pesquisas realizadas por tutoriais de programação, fornecido pelo *Google Trends*.

Costumam-se dividir linguagens de programação em duas categorias: de *baixo* e *alto nível*[10]. Linguagens de baixo nível são aquelas em que "as estruturas de controle e de dados refletem diretamente a arquitetura subjacente da máquina"[11]. Em contrapartida, as linguagens de alto nível são aquelas em que estas estruturas são determinadas pelo problema a ser resolvido [15]; ou seja, possuem maior nível de abstração.

Em 2000, Prechelt [59] comparou linguagens de programação divididas em três grupos:

- Linguagens de alta performance: C e C++, que são de mais baixo nível do que a maioria das linguagens de programação em uso atualmente. Tratam-se de linguagens *compiladas*, o que significa que seu código é transformado em código de máquina[12] para execução. Por esta razão, são utilizadas em projetos que requeiram alta performance e uso eficiente de memória, como sistemas operacionais [74, 77];

- Linguagem intermediária: Java, separada de C e C++ pela percepção popular de que sua performance é notavelmente inferior à de C e C++ para aplicações pesadas. Java se destaca por sua robustez, tendo se popularizado pelo fato de um mesmo código funcionar em diferentes máquinas [62];

- Linguagens de *scripting*: Perl, Python, Rexx e Tcl, que são de mais alto nível. Destas, a que mais se destaca é Python, pela simplicidade de sua sintaxe e o grande número de bibliotecas disponíveis para os mais diversos tipos de aplicação [72].

[10] *Low-level* e *high-level*, respectivamente.
[11] "[...] in which the control and data structures directly reflect the underlying machine architecture."
[12] Em essência, o código que o computador de fato utiliza para executar aplicações [15].

O autor comparou códigos desenvolvidos por programadores nas diferentes linguagens. O objetivo era resolver um problema que envolvia carregar um grande conjunto de dados na memória e depois retornar uma saída em formato de texto pesquisando pelos dados armazenados. Prechelt concluiu que, em geral, os programas em baixo nível alcançam seus objetivos mais rapidamente. Em contrapartida, os programas em alto nível requerem menos tempo e menos linhas de código para serem implementados. Em termos de produtividade (medida em linhas de código por hora), as linguagens de programação apresentam desempenhos similares. Prechelt reconheceu a simplicidade do problema proposto e admitiu duvidar "[...] que os resultados relativos para as linguagens de *script* seriam tão bons aplicados a outros problemas"[13].

Em artigo introduzindo a linguagem de programação Julia, os autores Bezanson et al. [10] resumem esta situação:

> O que o programador dinâmico perde em performance, o programador em C e em Fortran perde em produtividade. Um resultado infeliz [disso] é que as áreas mais desafiadoras da computação numérica tiveram os menores benefícios com o aumento da abstração e da produtividade oferecido por linguagens de alto nível. As consequências têm sido mais sérias do que muitos percebem.[14]

Portanto, o objetivo da linguagem é aproximar a comunidade científica do objetivo de ter uma linguagem de alto nível e de alta performance. Os resultados apresentados por Julia para o conjunto de problemas de teste apresentado em [10] são comparáveis aos de implementações em C e Fortran, além de mostrarem-se mais consistente do que os das demais linguagens testadas.

Para resolver problemas de otimização em Julia, pode-se utilizar o pacote JuMP [29], que permite a modelagem de problemas de vários tipos. Seu grande diferencial é que o pacote atua como um intermediário entre o programador e o *solver*, que é a ferramenta utilizada para de fato resolver os problemas. A forma como se adicionam restrições a um problema de otimização, por exemplo, são radicalmente diferentes entre os *solvers* Gurobi [40] e GLPK [53], de modo que a comparação entre *solvers* se torna um processo muito trabalhoso, mesmo para modelos pequenos. O JuMP resolve este problema com uma

[13] "[...] that the relative results for the script languages group would hold up well when applied to other problems."

[14] "As much as the dynamic language programmer misses out on performance, though, the C and Fortran programmer misses out on productivity. An unfortunate outcome is that the most challenging areas of numerical computing have benefited the least from the increased abstraction and productivity offered by higher-level languages. The consequences have been more serious than many realize."

forma unificada de modelar problemas, podendo traduzi-los automaticamente de *solver* para *solver*.

Exemplo 4.1. Considere o problema de otimização extraído de [32]:

$$\begin{aligned} \max \quad & 2x_1 + 3x_2 \\ \text{s.a} \quad & -4x_1 + 3x_2 \leq 12, \\ & 2x_1 + x_2 \leq 6, \\ & x_1 + x_2 \geq 3, \\ & 5x_1 + x_2 \geq 4, \\ & x_1, x_2 \geq 0. \end{aligned} \tag{33}$$

Este problema pode ser modelado em JuMP conforme o Código 1.

Código 1 Código em Julia do exemplo 3-6-2 usando o `JuMP.jl`.

```
using JuMP
using GLPK

lp = Model(GLPK.Optimizer)
@variable(lp, x[1:2] ≥ 0)
@objective(lp, Max, 2x[1] + 3x[2])
@constraint(lp, -4x[1] + 3x[2] ≤ 12)
@constraint(lp, 2x[1] + x[2] ≤ 6)
@constraint(lp, x[1] + x[2] ≥ 3)
@constraint(lp, 5x[1] + x[2] ≥ 4)
optimize!(lp)
println(value.(lp[:x]))
print(objective_value(lp))
```

A linha `lp = Model(GLPK.Optimizer)` especifica que o *solver* desejado é o GLPK. Se, por exemplo, for desejável alterar o *solver* para o Gurobi, é possível fazê-lo com `set_optimizer(lp, Gurobi.Optimizer)`. A chamada `optimize!(lp)` solicita que o *solver* escolhido resolva o problema. A Saída 2 confirma que a resolução foi bem-sucedida.

Saída 2 Saída do exemplo 3-6-2 usando o `JuMP.jl`.

```
[0.6000000000000003, 4.8]
15.6
```

Por todas as razões apresentadas, escolheu-se o Julia para implementar todos os códigos deste trabalho.

4.2. Técnicas de otimização inteira

Problemas de otimização inteira são de difícil resolução. É necessário utilizar ferramentas capazes de misturar diversas técnicas para resolvê-los. Tais ferramentas são conhecidas como *solvers*[15].

4.2.1. Técnicas exatas

Em geral, os *solvers* tentam resolver problemas de otimização inteira utilizando técnicas exatas de resolução. Estas técnicas são preferíveis por conduzirem o problema à solução ótima de fato. Em contrapartida, sua aplicação costuma tornar a resolução dos problemas mais lenta. O contrário de tais técnicas são as *heurísticas* e *meta-heurísticas*, abordadas adiante.

Geralmente, técnicas exatas envolvem resolver versões relaxadas dos problemas de otimização. Em seguida, verifica-se a factibilidade da resolução relaxada e, se necessário, dividem-se os problemas em problemas menores. Repete-se este processo iterativamente até obter-se uma solução que satisfaça as restrições inteiras e seja ótima.

Bradley, Hax e Magnanti [12] e Mohamed [56] comparam problemas de otimização inteira com suas versões relaxadas. É um erro assumir que a solução inteira ótima para um problema pode ser encontrada arredondando-se ou truncando-se o valor do caso relaxado. Isso se deve a dois motivos: a possibilidade de o valor obtido não ser factível e o fato de a solução inteira poder estar radicalmente distante da linear. Para resolver problemas assim, estes autores introduzem a técnica de *branch-and-bound*, resumida a seguir:

(a) Resolve-se a versão relaxada da formulação inicial do problema, P_0. Para-se se nenhuma restrição inteira for violada;

(b) Escolhe-se uma variável x que deveria ser inteira, mas tem valor $x = \alpha \notin \mathbb{Z}$, e criam-se dois novos problemas lineares: P_1, com restrição $x \leq \lfloor \alpha \rfloor$, e P_2, com restrição $x \geq \lceil \alpha \rceil$;

(c) Resolve-se um dos problemas não resolvidos. Se a solução não for a ótima, repete-se (b). Do contrário, para-se.

[15] Uma tradução literal seria "resolvedor".

O resultado deste procedimento é a criação de uma árvore, em que cada nó é um problema e cada folha é um problema a resolver [40]. A decisão sobre qual folha explorar e a estratégia usada para determinar mais rapidamente uma solução ótima dependem da implementação do *solver* [22].

Uma técnica mais robusta é o *branch-and-cut*. Esta ténica amplia o *branch-and-bound* por meio de *planos de corte*, que são restrições que reduzem o tamanho do poliedro do problema sem excluir soluções inteiras [17]. Quanto maior o número de soluções inteiras em que a restrição for ativa, melhor será o plano de corte. Como o poliedro resultante é menor do que o original, é provável que sejam necessárias menos iterações de *branch-and-bound* para resolver o problema.

Um exemplo de plano de corte é o de Gomory. Suponha que a i-ésima restrição linear de um problema com n variáveis de decisão seja $\sum_{j=1}^{n} a_{ij}x_j = b_i$. Então, o seu plano de corte de Gomory será a restrição original menos a restrição com todas as constantes arredondadas para baixo [50, 61]:

$$\sum_{j=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)x_i \leq b_i - \lfloor b_i \rfloor. \tag{34}$$

Métodos ainda mais robustos podem ser utilizados em busca de soluções exatas. Um exemplo é o *branch-and-price*, que adiciona a técnica de geração de colunas ao *branch-and-bound* e envolve resolver um problema principal e um subproblema, que espera-se que reduza o caminho trilhado para se obter a melhor solução. Este método pode ser combinado com o *branch-and-cut*, formando o que é conhecido como *branch-and-cut-and-price* [21, 76].

4.2.2. Técnicas heurísticas

Heurísticas são métodos aplicados a problemas específicos que podem gerar atalhos em direção à solução ótima. Estes métodos não têm o rigor matemático de algoritmos e, portanto, não existe a garantia de que sua aplicação será eficiente [15, 76].

Um exemplo de heurística para resolução do PCV é apresentado por Schermer [66]. O procedimento é resumido a seguir.

(a) Dado um PCV assimétrico com grafo $G = (\mathcal{V}, \mathcal{A})$, em que $\mathcal{V}$ é o conjunto de vértices e $\mathcal{A}$ é o conjunto de arestas do problema, gerar o seguinte modelo:

$$\min \sum_{i,j \in \mathcal{V}} x_{ij}c_{ij} \tag{35}$$

$$\text{s.a} \sum_{j \in \mathcal{V}} x_{ij} = 1, \qquad \forall i \in \mathcal{V},\ i \neq j, \tag{36}$$

$$\sum_{i \in \mathcal{V}} x_{ij} = 1, \qquad \forall j \in \mathcal{V},\ j \neq i, \tag{37}$$

$$x_{ii} = 0, \qquad \forall i \in \mathcal{V}, \tag{38}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in \mathcal{V}; \tag{39}$$

(b) Solicitar que o *solver* resolva o modelo atual do problema;

(c) Procurar pelo menor ciclo na solução gerada pelo *solver*, com conjunto S^* de vértices, e parar se $S^* = \mathcal{V}$;

(d) Adicionar a restrição $\sum_{i,j \in S^*} x_{ij} \leq |S^*| - 1, \forall S^* \subsetneq \mathcal{V}$ ao modelo. Voltar para (b).

Segundo Schermer, "prevenir apenas o menor ciclo geralmente é o suficiente para quebrar outros subciclos e mantém o modelo pequeno".[16] Duas maneiras são apresentadas para implementar esta heurística:

- Iterativa: Espera-se que o *solver* finalize sua resolução do problema para procurar pelo menor subciclo gerado. Então, gera-se um novo modelo e solicita-se que o *solver* o resolva desde o início;
- Com *lazy constraints*[17]: Inclui-se no modelo uma função de *callback*[18] que recebe atualizações de *status* do *solver*. Sempre que o *solver* encontra uma solução inteira ótima para o modelo fornecido, esta função procura pelo menor subciclo e o introduz dinamicamente no modelo.

Espera-se que o método de *lazy constraints* seja mais eficiente, pois o *solver* retém as informações obtidas sobre o modelo até o momento. Curiosamente, Schermer observou o contrário no teste que realizou para seu modelo de exemplo, demonstrando a natureza probabilística da aplicação de heurísticas ao problema.

Para PRVs, heurísticas mais robustas são necessárias, pois as violações de restrições não são tão óbvias quanto ciclos não-hamiltonianos. Um exemplo é o problema capacitado, que envolve otimizar o trajeto de uma frota de veículos com capacidade máxima

[16] "[...] preventing just the shortest cycle is often sufficient for breaking other subtours and will keep the model size smaller."

[17] Literalmente, "restrições preguiçosas". Estas restrições recebem este nome por passarem a atuar no problema apenas a partir do momento em que são violadas.

[18] Uma função f é função de *callback* quando é passada como argumento para uma função F, que, em algum momento, chama f durante sua execução.

de entregas para cada veículo. Além de procurar por ciclos que não incluem o depósito, Achuthan, Caccetta e Hill [2] propõem uma heurística que faz três pesquisas em subconjuntos de vértices. A primeira procura por violações a restrições dentro de ciclos envolvendo o depósito, a segunda procura por violações fora de cada ciclo e a terceira faz uma checagem final de todos os ciclos para os quais não foram encontradas restrições.

Existem técnicas mais gerais, conhecidas como meta-heurísticas, que podem ser aplicadas a diferentes tipos de problemas e conseguem evitar alguns dos obstáculos enfrentados pelas heurísticas. Exemplos populares já utilizados para resolver PRVs incluem a busca tabu, a colônia de formigas e os algoritmos evolucionários [76].

4.3. Comparações computacionais

Nesta seção, apresenta-se um método para comparar *solvers* e formulações de roteirização.

4.3.1. Comparação de formulações

O objetivo desta seção é demonstrar uma possível comparação de formulações de problemas. Toma-se como base o PCV, devido à maior simplicidade de implementação. No entanto, esta comparação também pode ser feita para diferentes formulações de VRPs.

Em 2021, Bazrafshan, Zolfani e Mirzapour Al-e-hashem [6] compararam as três formulações de PCV apresentadas na Seção 3.3. Os autores analisaram os desempenhos das formulações para 10 problemas diferentes, de 10 a 500 pontos, com coordenadas aleatórias para os vértices. Das três formulações, os autores constataram que a GG é a melhor, seguida pela MTZ. A partir de 19 pontos, a formulação DFJ sequer consegue resolver problemas no tempo limite.

A família de restrições da DFJ é imposta sobre todos os subconjuntos próprios do problema, exceto pelo conjunto vazio. Isso significa que o número de restrições desta família é $|2^{\mathcal{V}}| - 2$, sendo $2^{\mathcal{V}}$ o conjunto potência do conjunto de vértices do problema, $\mathcal{V}$. Este crescimento exponencial sobrecarrega o *solver* logo no início da resolução do problema, inviabilizando-a.

Para resolver os problemas relaxados, a formulação MTZ leva vantagem sobre a GG, por adicionar menos variáveis e restrições ao problema. Ainda assim, os autores consideram a GG melhor, pois suas variáveis de ordenação são reais, ao passo que as da MTZ são inteiras. Como resultado, a MTZ tem desempenho inferior quando o modelo é passado da forma relaxada para a forma inteira.

Em relação às formulações do PCV, esta seção tem dois objetivos: verificar se as conclusões de Bazrafshan, Zolfani e Mirzapour Al-e-hashem [6] sobre as formulações GG e

MTZ se confirmam e comparar estas formulações à DFJ com a técnica de *lazy constraints* apresentada na Seção 4.2.2. Para tal, foram criados 3 conjuntos com 10 problemas gerados aleatoriamente em cada um. Os problemas do conjunto A têm 10 pontos; do B, têm 15; e do C têm 20. Em seguida, os problemas foram resolvidos por três *solvers*: GLPK [53], CPLEX [24] e Gurobi [40]. Para estes testes, foi utilizado um *IdeaPad* com processador Intel Core i5-10210U, 1.60 GHz, memória de 8 GB e suporte a até 8 *threads*. Os tempos de resolução foram fornecidos pelos próprios *solvers*, gerando a Tabela 2. As implementações são fornecidas no Apêndice B.1.

Constata-se que a formulação GG de fato é superior à MTZ para a resolução completa do PCV. Ademais, a formulação DFJ com *lazy constraints* consegue chegar à resposta ótima mais rapidamente do que as demais em todos os testes exceto um. Talvez esta anomalia possa ser explicada por se tratar do primeiro teste realizado com cada *solver*. É possível que os *solvers* contabilizem algum tempo de inicialização adicional nesta situação.

4.3.2. Comparação de solvers

Para comparar os *solvers*, utilizou-se a metodologia de Dolan e Moré [28]. Sejam $\mathcal{P}$ e $\mathcal{S}$ conjuntos de problemas e *solvers*, respectivamente. Para todo problema $p \in \mathcal{P}$ e *solver* $s \in \mathcal{S}$, obtém-se uma performance $t_{p,s}$ de acordo com alguma métrica de desempenho (tempo de resolução, número de funções utilizadas, etc.). Então, obtêm-se as razões entre os desempenhos de cada *solver* e o desempenho do melhor *solver* para determinado problema:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} \mid s \in \mathcal{S}\}}. \tag{40}$$

Então, define-se uma função estatística chamada de *performance profile*[19], que representa uma distribuição das probabilidades de um problema $p \in \mathcal{P}$ ser resolvido por um *solver* $s \in \mathcal{S}$ dentro de um intervalo $[1, \tau]$ de razões em relação ao resultado do melhor *solver* para o problema em questão:

$$\rho_s(\tau) = \frac{1}{|\mathcal{P}|}|\{p \in \mathcal{P} \ \mid r_{p,s} \leq \tau\}|. \tag{41}$$

Aplicando este procedimento aos resultados da Tabela 2, obtêm-se as Figuras 8 a 10, onde são comparados os *performance profiles* dos três *solvers* testados, tomando o DFJ com *lazy constraints* como base.

[19] "Perfil de performance."

Tabela 2: Tempos (s) de resolução para diferentes *solvers* e formulações.

	GLPK			CPLEX			Gurobi		
	DFJ*	MTZ	GG	DFJ*	MTZ	GG	DFJ*	MTZ	GG
A1	2.964	0.746	**0.055**	0.703	0.461	**0.046**	0.641	0.333	**0.022**
A2	**0.002**	0.228	0.056	**0.008**	0.118	0.050	**0.008**	0.191	0.021
A3	**0.003**	0.087	0.036	**0.011**	0.077	0.066	**0.006**	0.116	0.016
A4	**0.002**	0.016	0.035	**0.005**	0.026	0.025	**0.003**	0.060	0.035
A5	**0.002**	0.220	0.217	**0.005**	0.093	0.039	**0.008**	0.149	0.017
A6	**0.002**	0.032	0.017	**0.008**	0.031	0.015	**0.006**	0.043	0.015
A7	**0.007**	–	0.057	**0.012**	0.078	0.044	**0.008**	0.085	0.035
A8	**0.002**	1.188	0.177	**0.005**	1.356	0.140	**0.004**	0.488	0.059
A9	**0.004**	0.074	0.089	**0.007**	0.031	0.036	**0.009**	0.091	0.021
A10	**0.003**	0.210	0.112	**0.013**	0.049	0.057	**0.007**	0.178	0.031
B1	**0.048**	3.341	–	**0.035**	0.354	0.514	**0.038**	0.274	0.208
B2	**0.015**	1.412	1.047	**0.061**	0.139	0.377	**0.044**	0.371	0.159
B3	**0.053**	–	18.238	**0.035**	0.215	0.183	**0.025**	0.225	0.078
B4	**0.028**	47.536	0.861	**0.037**	18.583	0.218	**0.019**	3.612	0.091
B5	**0.022**	1.244	0.462	**0.018**	0.338	0.114	**0.019**	0.293	0.063
B6	**0.006**	0.274	3.103	**0.036**	0.074	0.098	**0.026**	0.140	0.069
B7	**0.008**	–	3.711	**0.028**	0.332	0.138	**0.012**	0.261	0.134
B8	**0.022**	2.148	3.205	**0.016**	0.540	0.276	**0.012**	0.364	0.107
B9	**0.010**	–	18.597	**0.016**	0.091	0.087	**0.011**	0.119	0.063
B10	**0.019**	–	43.736	**0.036**	18.869	0.192	**0.019**	2.609	0.094
C1	**0.022**	–	–	**0.061**	2.832	0.127	**0.055**	1.485	0.304
C2	**0.064**	9.668	8.949	**0.149**	0.408	0.372	**0.046**	1.434	0.135
C3	**0.012**	11.405	–	**0.026**	0.484	0.501	**0.026**	0.372	0.150
C4	**0.226**	–	–	**0.116**	8.615	0.428	**0.054**	2.957	0.229
C5	**0.035**	27.043	22.634	**0.040**	8.298	0.138	**0.021**	3.103	0.241
C6	**0.023**	–	–	**0.085**	0.732	1.192	**0.059**	1.670	0.234
C7	**0.160**	13.547	–	**0.049**	1.304	0.531	**0.046**	0.918	0.271
C8	**0.089**	–	18.351	**0.137**	7.881	0.468	**0.056**	4.076	0.289
C9	**0.138**	31.818	–	**0.075**	4.832	0.225	**0.052**	2.992	0.130
C10	**0.074**	1.840	–	**0.057**	0.117	0.271	**0.024**	0.244	0.110

A, conjuntos de 10 pontos; B, conjuntos de 15 pontos; C, conjuntos de 20 pontos; DFJ*, DFJ com *lazy constraints*. Tempos maiores do que 60 s foram desconsiderados e substituídos por travessão.

Figura 8: *Performance profiles* para o conjunto de 10 pontos

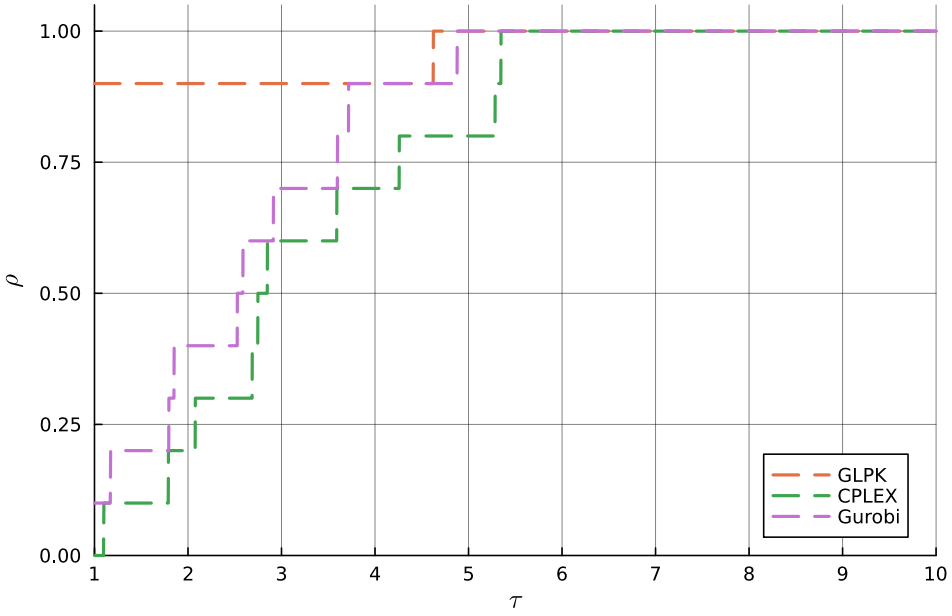


Figura 9: *Performance profiles* para o conjunto de 15 pontos

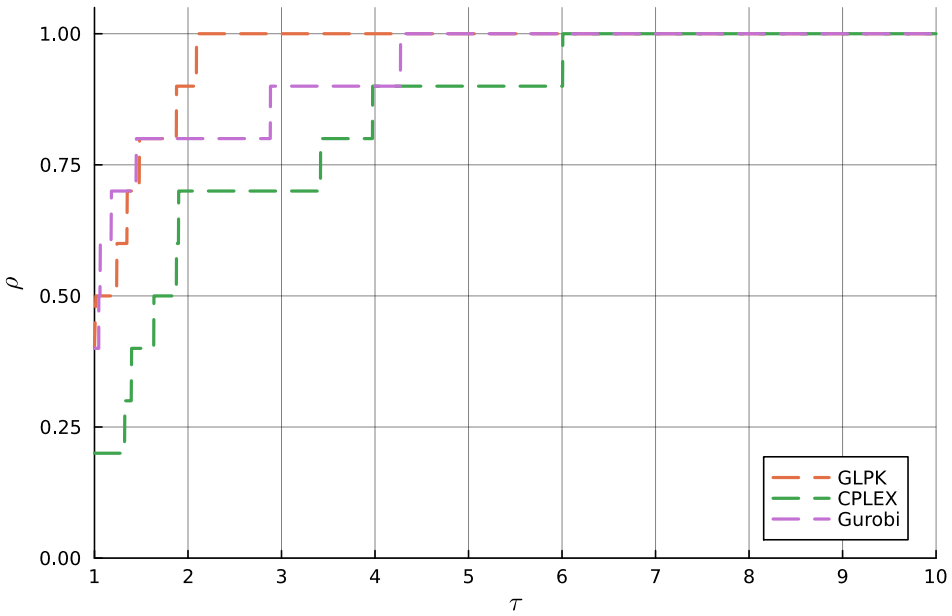
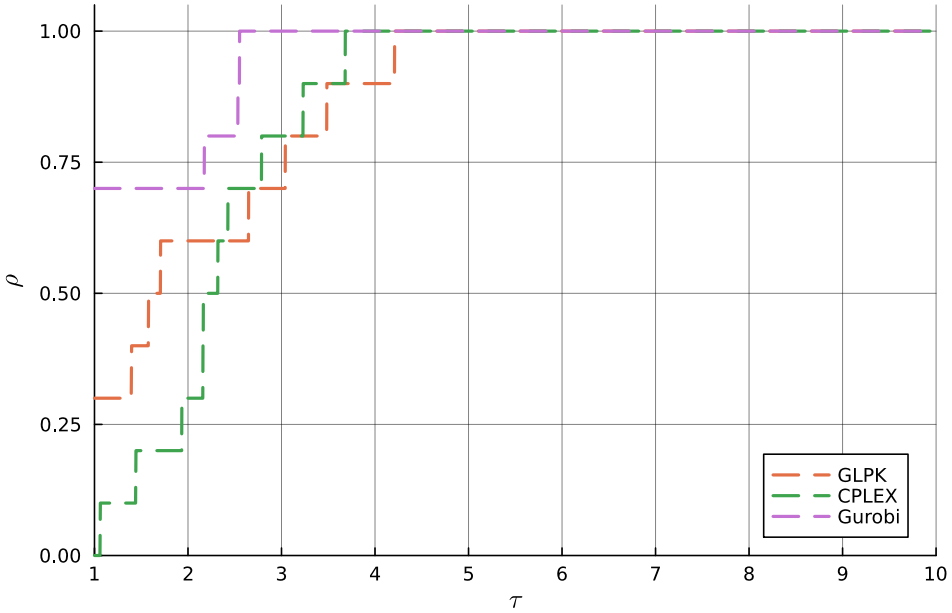


Figura 10: *Performance profiles* para o conjunto de 20 pontos



Cada "degrau" num gráfico de *performance profiles* indica a razão de problemas resolvidos no tempo relativo indicado pelas abscissas. Assim, o *solver* cujo *performance profile* intercepta as ordenadas com valor mais alto é o que chega à solução ótima mais rapidamente na maioria dos casos. Para 10 pontos, o GLPK é dominante neste aspecto. O Gurobi passa a dominar conforme o número de pontos aumenta.

Com 20 pontos, o Gurobi começa a se distanciar dos demais *solvers*. O GLPK e o CPLEX apresentam desempenhos similares. Realizou-se mais um teste, este com 10 problemas de 50 pontos, para verificar a tendência de desempenho dos *solvers* em situações mais extremas. Os resultados são apresentados na Tabela 3.

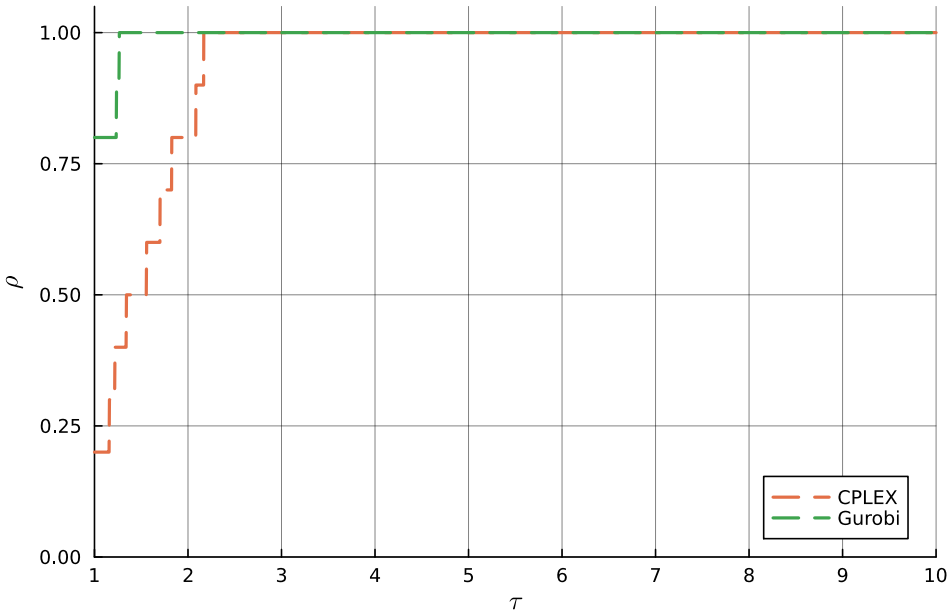
A Figura 11 mostra uma comparação dos *performance profiles* para 50 pontos, desconsiderando o GLPK. Verifica-se que o Gurobi continua à frente, apesar de o CPLEX conseguir uma performance comparável, sendo pouco mais que duas vezes mais lento no pior dos casos.

Tabela 3: Tempos (s) de resolução para PCVs de 50 pontos usando DFJ com *lazy constraints*

GLPK	CPLEX	Gurobi
–	1.530	**0.983**
13.778	**0.312**	0.395
–	0.652	**0.535**
–	**0.365**	0.451
–	0.598	**0.517**
–	0.918	**0.440**
–	0.511	**0.381**
–	1.523	**0.702**
–	1.071	**0.630**
35.082	0.747	**0.410**

Tempos maiores do que 60 s foram substituídos por travessão.

Figura 11: *Performance profiles* para o conjunto de 50 pontos



5. Problemas de roteamento de veículos com janelas de tempo

O objetivo desta seção é introduzir problemas de roteamento de veículos (PRVs), que são generalizações do PCV para frotas [26]. Existem vários tipos de PRVs, cada qual com suas próprias famílias de restrições [73, 76]. Uma consequência desta variedade é que a complexidade computacional dos PRVs é maior do que a do PCV, fato que se reflete no empenho muito maior da comunidade científica em encontrar soluções exatas para o PCV do que para PRVs [48].

O PRV com janelas de tempo (PRVJT) considera que cada ponto tem uma janela de tempo específica para ser atendido por um veículo. Como o PRVJT não considera o nível de combustível dos veículos, ele é mais apropriado para criar modelos básicos para ônibus a combustão, cujo combustível dura mais e pode ser reabastecido rapidamente. As janelas de tempo de cada região a ser atendida podem ser definidas conforme o planejamento urbano da cidade.

5.1. Modelo genérico

Vieira [76] apresenta o seguinte modelo de PRVJT, que é uma extensão de um problema capacitado (PRVC), em que os veículos têm uma capacidade máxima de carga e cada ponto tem sua própria demanda. Uma implementação básica deste modelo pode ser encontrada no Apêndice B.2.

$$\min \sum_{\substack{i,j\in\mathcal{V}\\k\in K}} c_{ij}x_{ijk} \tag{42}$$

$$\text{s.a} \sum_{\substack{k\in K\\j\in\mathcal{V}^*}} x_{0jk} \leq |K|, \tag{43}$$

$$\sum_{j\in\mathcal{V}^*} x_{0jk} = \sum_{j\in\mathcal{V}^*} x_{j0k} \leq 1, \qquad \forall k \in K, \tag{44}$$

$$\sum_{\substack{k\in K\\j\in\mathcal{V}}} x_{ijk} = 1, \qquad \forall i \in \mathcal{V}^*, \tag{45}$$

$$\sum_{j\in\mathcal{V}} x_{ijk} - \sum_{j\in\mathcal{V}} x_{jik} = 0, \qquad \forall k \in K,\ i \in \mathcal{V}^*, \tag{46}$$

$$\sum_{\substack{i,j\in S\\k\in K}} x_{ijk} \leq |S| - v(S), \qquad \forall S \subseteq \mathcal{V}^*,\ |S| \geq 2, \tag{47}$$

$$\sum_{\substack{k\in K\\i\in\mathcal{V}\\i\neq j}} x_{ijk}(b_i + s_i + t_{ij}) \leq b_j, \qquad \forall j \in \mathcal{V}^*, \tag{48}$$

$$e_i \leq b_i \leq l_i, \qquad \forall i \in \mathcal{V}, \tag{49}$$

$$x_{ijk} \in \{0, 1\}, \qquad \forall i, j \in \mathcal{V},\ k \in K. \tag{50}$$

Esta formulação considera o problema como um grafo $G = (\mathcal{V}, \mathcal{A})$. O depósito é o vértice 0, e $\mathcal{V}^* = \mathcal{V}\backslash\{0\}$. Considera-se uma frota K de veículos homogêneos de capacidade Q, e para cada ponto existe uma variável m_i indicando sua demanda. As variáveis e_i e l_i indicam o início e o fim do intervalo de atendimento a cada ponto, inclusive do depósito. A variável b_i indica o horário em que um veículo chega ao ponto i; s_i indica o tempo de atendimento ao ponto i; t_{ij} indica o tempo de deslocamento necessário entre i e j.

A Expressão (42) define o problema de minimização de distâncias. Usam-se $|K|$ matrizes de adjacência, uma para cada veículo. A Restrição (43) garante que não saiam do depósito mais veículos do que a frota comporta. Também é possível garantir que todos os veículos saiam do depósito, transformando a inequação em equação, ou considerando o depósito como sendo dois vértices distintos, 0 e $n + 1$. Neste caso, os veículos que não forem utilizados farão ciclo entre 0 e $n + 1$ [68]. As Restrições (44) asseguram que todos os veículos saiam e retornem ao depósito. As Restrições (45) faz com que apenas um veículo atenda cada ponto. As Restrições (46) garantem que os veículos saiam de todos os pontos nos quais entrarem.

As Restrições (47) impedem que sejam formados ciclos que violem as capacidades dos veículos. A função $v(S)$ retorna o número mínimo de veículos necessários para atender o conjunto de pontos. É possível defini-la como

$$v(S) = \left\lceil \sum_{i\in S} \frac{m_i}{Q} \right\rceil. \tag{51}$$

Supondo um conjunto de n pontos, se m veículos forem necessários para atendê-los, então restam a estes veículos $n - m$ pontos para os quais se deslocar. Quaisquer deslocamentos a mais entre os pontos de S denunciam violações às restrições de capacidade dos veículos.

As Restrições (48) asseguram que o veículo que vai de i a j finalizará o seu atendimento a i a tempo de chegar a j no horário de atendimento. As Restrições (49) garantem

que todos os veículos cheguem aos pontos de entrega dentro da janela de tempo definida para cada ponto. Por fim, as Restrições (50) garantem que as variáveis de decisão sejam todas inteiras.

As Restrições (48) têm comportamento não-linear, pois tanto x_{ijk} quanto b_i são variáveis de decisão. Cordeau et al. [23] e Kallehauge et al. [45] propõem linearizar esta expressão trocando-a por outra da forma

$$b_i + s_i + t_{ij} - M_{ij}(1 - x_{ijk}) \leq b_j, \ \forall i, j \in \mathcal{V}, \forall k \in K, \tag{52}$$

onde M_{ij} é uma constante cujo valor precisa ser, no mínimo, $\max\{l_i + t_{ij} - e_j\}$, $(i, j) \in \mathcal{A}$.

É possível utilizar as restrições de capacidade para definir um número máximo de pontos que cada ônibus possa atender. Se capacidades não forem uma preocupação, é possível escolher $Q \to \infty$.

5.2. Modelo escolar com rotas pré-determinadas

Problemas em uma mesma categoria podem ter formulações radicalmente diferentes, a depender das considerações matemáticas tecidas. Fügenschuh, Martin e Stöveken [35], por exemplo, propõem um modelo de PRVJT que permite a roteirização de ônibus escolares que também podem ser usufruídos pela população geral. Este modelo considera $\mathcal{S}$ o conjunto de escolas a serem atendidas. Para toda escola $s \in \mathcal{S}$, é definida uma janela de tempo de atendimento $[\underline{\tau}_s, \overline{\tau}_s]$, com $\underline{\tau}_s, \overline{\tau}_s \in \mathbb{Z}_+$.

Esta formulação define como *viagens*[20] as sequências de pontos pelas quais cada ônibus deve passar até chegar a cada escola. Trabalha-se com um conjunto já definido de viagens, $\mathcal{V}$, com cada viagem individual sendo representada por $t \in \mathcal{V}$, e cada aresta que conecta os pontos das viagens estando contida em $\mathcal{A}$. As constantes $\underline{\alpha}_t$ e $\overline{\alpha}_t$, com $\underline{\alpha}_t \leq \overline{\alpha}_t$ e $\underline{\alpha}_t, \overline{\alpha}_t \in \mathbb{Z}_+$, definem o intervalo aceitável de valores para α_t, que indica quando a viagem t é iniciada. O tempo entre o início e o fim de uma viagem é $\delta_t^{\text{trip}} \in \mathbb{Z}_+$.

$\mathcal{P}$ é definido como o conjunto das tuplas (s, t) que associam uma viagem t a uma escola s. O tempo entre o início e o fim de uma viagem até uma escola é $\delta_t^{\text{school}} \in \mathbb{Z}_+$. O intervalo $[\underline{\omega}_{st}^{\text{school}}, \overline{\omega}_{st}^{\text{school}}]$, com $\underline{\omega}_{st}^{\text{school}}, \overline{\omega}_{st}^{\text{school}} \in \mathbb{Z}_+$, define quanto tempo os alunos podem esperar pelo ônibus.

É possível que um aluno precise passar por mais do que uma viagem para chegar à sua escola. Neste caso, considera-se o conjunto $\mathcal{C}$ de tuplas (t_1, t_2), que representam uma *troca*[21] da viagem t_1 para a viagem t_2. A viagem t_1 é chamada de *feeder*, e t_2 é chamada

[20] "*Trips*".
[21] "*Change*".

de *collector*. O tempo decorrido entre o começo de uma viagem *feeder* e a chegada até o ponto de troca é $\delta^{\text{feeder}}_{t_1t_2} \in \mathbb{Z}_+$. Já o tempo que o ônibus que coleta os alunos deixados leva para chegar a este ponto é $\delta^{\text{collector}}_{t_1t_2} \in \mathbb{Z}_+$. O intervalo de tempo aceitável de espera no ponto que conecta as duas viagens é $[\underline{\omega}^{\text{change}}_{t_1t_2}, \overline{\omega}^{\text{change}}_{t_1t_2}]$, com $\underline{\omega}^{\text{change}}_{t_1t_2}, \overline{\omega}^{\text{change}}_{t_1t_2} \in \mathbb{Z}_+$.

Ônibus que tiverem terminado suas viagens retornam ao depósito. A conexão entre viagens servidas por um mesmo ônibus é chamada de *bloco*. O deslocamento entre o último ponto de uma viagem t_1 e o primeiro ponto de uma viagem t_2 são chamados de *shifts*. O tempo necessário para completar um *shift* é $\delta^{\text{shift}}_{t_1t_2} \in \mathbb{Z}_+$.

A cada viagem são associadas variáveis v_t e w_t, sendo $v_t = 1$ se t for a primeira viagem em um bloco, ou 0 do contrário. A variável w_t é igual a 1 se t for a última viagem do bloco, ou 0 do contrário. Também consideram-se variáveis $x_{t_1t_2} \in \{0, 1\}$, com $x_{t_1t_2} = 1$ apenas se um ônibus servir t_2 após servir t_1.

O modelo proposto pelos autores é:

$$\min \sum_{t \in \mathcal{V}} C_t v_t + \sum_{(t_1,t_2) \in \mathcal{A}} \delta^{\text{shift}}_{t_1t_2} x_{t_1t_2}, \tag{53}$$

$$\text{s.a} \sum_{(t_1,t_2) \in \mathcal{A}} x_{t_1t_2} + v_{t_2} = 1, \qquad \forall t_2 \in \mathcal{V}, \tag{54}$$

$$\sum_{(t_1,t_2) \in \mathcal{A}} x_{t_1t_2} + w_{t_1} = 1, \qquad \forall t_1 \in \mathcal{V}, \tag{55}$$

$$\alpha_{t_1} + \delta^{\text{trip}}_{t_1} + \delta^{\text{shift}}_{t_1t_2} - M(1 - x_{t_1t_2}) \leq \alpha_{t_2}, \qquad \forall (t_1, t_2) \in \mathcal{A}, \tag{56}$$

$$\alpha_{t_1} + \delta^{\text{feeder}}_{t_1t_2} + \underline{\omega}_{t_1t_2} \leq \alpha_{t_2} + \delta^{\text{collector}}_{t_1t_2}, \qquad \forall (t_1, t_2) \in \mathcal{C}, \tag{57}$$

$$\alpha_{t_1} + \delta^{\text{feeder}}_{t_1t_2} + \overline{\omega}_{t_1t_2} \geq \alpha_{t_2} + \delta^{\text{collector}}_{t_1t_2}, \qquad \forall (t_1, t_2) \in \mathcal{C}, \tag{58}$$

$$\alpha_t + \delta^{\text{school}}_{st} + \underline{\omega}^{\text{school}}_{st} \leq \underline{\tau}_s + 5\tau_s, \qquad \forall (s, t) \in \mathcal{P}, \tag{59}$$

$$\alpha_t + \delta^{\text{school}}_{st} + \overline{\omega}^{\text{school}}_{st} \geq \underline{\tau}_s + 5\tau_s, \qquad \forall (s, t) \in \mathcal{P}. \tag{60}$$

A Expressão (53) indica que este é um problema de minimização biobjetivo. Deseja-se minimizar as distâncias percorridas e o tempo de *shift* dos ônibus que servirem blocos. As Restrições (54) asseguram que cada viagem será ou a primeira realizada por um ônibus, ou estará conectada a uma viagem anterior. Por outro lado, as Restrições (55) garantem que cada viagem seja a última, ou que seja antecessora de outra.

As Restrições (56) garantem que todos os ônibus que conectam viagens t_1 e t_2 completem t_1 a tempo de começarem t_2. As Restrições (57) e (58) forçam os ônibus de viagens do tipo *collector* a chegarem aos pontos de coleta dentro do intervalo aceitável

de espera dos alunos. As Restrições (59) e (60) definem os tempos de início das aulas para cada escola[22]. Neste caso, os autores consideraram que os tempos aceitáveis sejam separados por intervalos de 5 minutos, com

$$\tau_s = \frac{\overline{\tau_s} - \underline{\tau_s}}{5}. \tag{61}$$

Posteriormente, Fügenschuh [34] apresentou uma versão expandida deste modelo, com testes de caso para escolas alemãs. Constatou-se que seria possível diminuir de 10 a 25% a frota de ônibus em operação nos locais selecionados.

5.3. Modelo de frota mista

As dificuldades associadas à aquisição de ônibus elétricos destacadas na Seção 2.2 tornam uma modelagem mista interessante. Nesta seção, é apresentado um modelo de *Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and recharging stations* (*E-FSMFTW*)[23] proposto por Hiermann et al. [42].

A notação dos autores considera que C é o conjunto de pontos a serem atendidos e F é o conjunto das estações de recarga. Existe um único depósito e um conjunto V de tipos de veículos. Considera-se o conjunto $N = C \cup F'$, onde F' é um conjunto de cópias dos pontos de F. Este conjunto é criado para que vários ônibus elétricos possam acessar uma mesma estação de recarga. O depósito é representado pelo vértice de partida u_0 e o vértice de chegada u_{n+1}. Tem-se ainda $N_0 = N \cup \{u_0\}$, $N_{n+1} = N \cup \{u_{n+1}\}$ e $N_{0,n+1} = N_0 \cup N_{n+1}$. A mesma notação se aplica a C e F'. Além disso, existem variáveis $x_{ij}^k \in \{0, 1\}$, com $x_{ij}^k = 1$ apenas se um veículo do tipo $k \in V$ se desloca do ponto i ao ponto j.

As variáveis Q^k definem a capacidade máxima de veículos do tipo k. A cada ponto i estão associadas uma variável de demanda, p_i, e uma variável com a carga atual do veículo de tipo k que o atende, q_i^k. Da mesma forma, Y^k representa a capacidade energética máxima dos veículos do tipo k, com y_i^k representando a energia atual do veículo de tipo k que atende o ponto i. O consumo energético por unidade de distância percorrida por veículos do tipo k é r^k, e o tempo de recarga por unidade de energia é g^k.

Para definir janelas de tempo, os autores usam as variáveis e_i, l_i, s_i, t_{ij}, com os mesmos significados atribuídos por Vieira [76] na Seção 5.1, e τ_i, que representa o momento

[22] As Restrições (57) a (60) criam dependências entre as janelas de tempo, razão pela qual os autores classificam o modelo completo como um PRV de *coupled time windows* (CTW).

[23] Uma possível tradução para o português seria "PRVJT com tamanho de frota elétrica, veículos mistos e estações de recarga".

em que o veículo começa a atender o ponto i. Consideram-se ainda a variável d_{ij}, que expressa a distância do ponto i ao ponto j, o custo de aquisição de veículos do tipo k, representado por f^k, e o custo de deslocar um veículo do tipo k do ponto i ao ponto k, denotado por c_{ij}^k.

Com esta notação, os autores modelam o problema da seguinte maneira:

$$\min \sum_{\substack{k\in V\\ j\in N}} f^k x_{0j}^k + \sum_{\substack{k\in V\\ i\in N_0\\ j\in N_{n+1}\\ i\neq j}} c_{ij}^k x_{ij}^k \tag{62}$$

$$\text{s.a} \sum_{\substack{k\in V\\ j\in N_{n+1}\\ i\neq j}} x_{ij}^k = 1, \qquad \forall i \in C, \tag{63}$$

$$\sum_{\substack{k\in V\\ j\in N_{n+1}\\ i\neq j}} x_{ij}^k \leq 1, \qquad \forall i \in F', \tag{64}$$

$$\sum_{\substack{i\in N_{n+1}\\ i\neq j}} x_{ji}^k - \sum_{\substack{i\in N_0\\ i\neq j}} x_{ij}^k = 0, \qquad \forall j \in N, k \in V, \tag{65}$$

$$e_j \leq \tau_j \leq l_j, \qquad \forall j \in N_{0,n+1}, \tag{66}$$

$$\tau_i + (t_{ij} + s_i)x_{ij}^k - l_0(1 - x_{ij}^k) \leq \tau_j, \qquad \forall k \in V, i \in C_0, j \in N_{n+1}, i \neq j, \tag{67}$$

$$\tau_i + t_{ij}x_{ij}^k + g^k(Y^k - y_i^k) - (l_0 + g^k Y^k)(1 - x_{ij}^k) \leq \tau_j, \qquad \forall k \in V, i \in F', j \in N_{n+1}, i \neq j, \tag{68}$$

$$q_j^k \leq q_i^k - p_i x_{ij}^k + Q^k(1 - x_{ij}^k), \qquad \forall k \in V, i \in N_0, j \in N_{n+1}, i \neq j, \tag{69}$$

$$0 \leq q_j^k \leq Q^k, \qquad \forall k \in V, j \in N_{0,n+1}, \tag{70}$$

$$0 \leq y_j^k \leq y_i^k - (r^k d_{ij})x_{ij}^k + Y^k(1 - x_{ij}^k), \qquad \forall k \in V, i \in C, j \in N_{n+1}, i \neq j, \tag{71}$$

$$0 \leq y_j^k \leq Y^k - (r^k d_{ij})x_{ij}^k, \qquad \forall k \in V, i \in F_0', j \in N_{n+1}, i \neq j, \tag{72}$$

$$y_0^k = Y^k, \qquad \forall k \in V, \tag{73}$$

$$x_{ij}^k \in \{0, 1\}, \qquad \forall i \in N_0, j \in N_{n+1}, i \neq j, k \in V. \tag{74}$$

A Expressão (62) estabelece um problema biobjetivo. O primeiro objetivo é minimizar os gastos de aquisição dos veículos que sairão do depósito. O segundo objetivo é minimizar o custo total associado aos deslocamentos de cada tipo de veículo. As Restrições (63) forçam que apenas um veículo passe por cada ponto do problema. As Restrições (64) permitem que os veículos elétricos façam (ou não) recarga nas estações. As Restrições (65) asseguram que todos os veículos entrem e saiam de cada vértice, salvo o caso especial do depósito, que é representado por dois vértices.

As Restrições (66) e (67) são análogas às Restrições (48) e (52), com l_0 correspondendo a M. As Restrições (68) garantem que os veículos não gastem mais tempo recarregando suas baterias ao máximo do que o necessário para começar a atender o pró-

ximo ponto. A constante $l_0 \to \infty$ tenta garantir que esta família de restrições não afete veículos que não se deslocarem de uma estação de recarga i a um ponto de atendimento j. Como as unidades de energia usadas podem variar, é possível que $l_0 < g^k(Y^k - y_i^k)$. Para evitar que isso quebre o modelo, soma-se $g^k Y^k$ a l_0, pois $l_0 + g^k Y^k > g^k(Y^k - y_i^k)$.

As Restrições (69) garantem que as cargas dos veículos sejam atualizadas ao longo dos seus trajetos, enquanto as Restrições (70) asseguram que os veículos não façam viagens cuja demanda total exceda suas capacidades. As Restrições (71) e (72) funcionam analogamente, atualizando as energias dos veículos e garantindo que não seja utilizada mais energia do que a bateria do respectivo veículo permite. Por fim, as Restrições (73) asseguram que todos os veículos comecem a operar com o máximo de energia, e as Restrições (74) estabelecem que as variáveis de decisão dos deslocamentos são binárias.

Este modelo pode ser expandido para considerar recargas parciais. Isso é preferível, pois o ideal para baterias de lítio é que sua carga seja mantida entre 65 e 75% do valor máximo. Além disso, a não-linearidade de recarga das baterias pode ser contabilizada para garantir uso mais preciso das baterias e do tempo. Formulações mais aprofundadas, como as de Cataldo-Díaz, Linfati e Escobar [19], Hellmark [41], Zhang, Wang e Qu [79] e Zuo et al. [80], levam em conta tais fatores.

5.4. Comparação de modelos

Os três modelos apresentados nesta seção (genérico [76], escolar [35], frota mista [42]) demonstram a grande variabilidade das formulações de PRVJTs, advindas das considerações feitas em diferentes contextos. A seguir, estão listadas algumas diferenças e similaridades notáveis.

Função objetivo

O modelo [76] possui apenas um objetivo de minimização, enquanto [35, 42] possuem dois. Parece ser uma tendência dos modelos mais complexos serem pelo menos biobjetivos, dada a profundidade das suas considerações acerca dos veículos utilizados. Outros exemplos desta tendência incluem [19, 33, 47, 79, 80].

Variáveis de decisão

O modelo [35] se distingue dos demais por considerar rotas já planejadas. Assim, este modelo não se utiliza da ideia de matrizes de adjacência. O modelo [76] considera várias matrizes de adjacência, uma para cada veículo da frota, ao passo que [42] considera que

veículos distintos de um mesmo tipo compartilham da mesma matriz de adjacência, de modo similar à modelagem em [2]. Tendo em vista as considerações computacionais realizadas na Seção 4, esta última abordagem é preferível, pois economiza variáveis inteiras.

Janelas de tempo

Em [42, 76], as janelas de tempo são pré-definidas. Em [35], um dos desafios é definir quais são os horários ótimos para as janelas de tempo. Isso se deve ao fato de [35] abordar um problema de planejamento urbano, ao passo que [42, 76] são modelos pensados para entregas, cujos horários são controlados por fatores externos.

Garantia da factibilidade

Em [76], a factibilidade do modelo é garantida por restrições sobre todos os subconjuntos de pontos do problema, de maneira similar à formulação DFJ do PCV. Em [35], ela é garantida pelo ajuste mútuo das janelas de tempo. Já [42] assegura a factibilidade do modelo por meio de janelas de tempo, além de produzir cópias das estações de recarga do modelo para garantir que vários veículos possam passar pelo mesmo vértice. Como [35] trabalha com rotas já conhecidas, o modelo evita este aumento de complexidade computacional.

6. Conclusão

O principal objetivo deste trabalho era o estudo teórico dos problemas de roteamento de veículos capacitado e com janelas de tempo, incluindo aplicações. Para este propósito, diversos empenhos foram empregados, a começar pelo estudo de diferentes disciplinas do curso de matemática da UFSC Blumenau. O estudante já havia tido um primeiro contato com a linguagem Julia e resolução computacional de problemas matemáticos na disciplina de Métodos Numéricos (MAT1831), antes de iniciar a pesquisa. Em semestres posteriores, estudou Otimização Contínua (MAT1032) e Fundamentos da Matemática (MAT1121). Todas as disciplinas foram ministradas pelo orientador de pesquisa.

Concomitantemente, o estudante se dedicou à leitura de livros de álgebra linear e otimização linear, expondo-se ao aprendizado de ferramentas e técnicas matemáticas que estão além daquilo que é abordado em seu curso de engenharia. Para solidificar os conhecimentos adquiridos, implementou o método simplex em ambiente computacional [20] e aprendeu a realizar a modelagem de problemas de otimização utilizando Julia [10] e o pacote JuMP [29]. Ao mesmo tempo, documentou os resultados das leituras, implementações e testes computacionais neste relatório. Durante a condução da pesquisa, por diversas vezes este documento foi reescrito e reorganizado, visando concatenar da melhor maneira possível as ideias conducentes à compreensão de PRVs.

Pode-se afirmar que este objetivo foi cumprido, pois os procedimentos descritos levaram o estudante a compreensão satisfatória das técnicas, características e considerações acerca de modelos de PRVs, como demonstrado pela descrição de três modelos diferentes na Seção 5, além da citação de outros modelos analisados. Focou-se, principalmente, na problemática de ônibus, introduzida na Seção 2, como motivação para a realização deste estudo.

O estudo computacional do problema tornou possíveis as implementações de modelos de roteirização. Com isso, obteve-se êxito na resolução de diferentes formulações do PCV. Já para PRVs, este relatório se limitou à modelagem do PRVJT de Vieira [76], pois a resolução de tais problemas renderia uma pesquisa própria, como indicam os grandes esforços dos artigos analisados para definir e implementar heurísticas sofisticadas. Ainda assim, foi possível aprender o básico sobre estes métodos, o que foi crucial para a compreensão e comparação dos *solvers* testados.

Pode-se dizer que os principais objetivos desta pesquisa foram concluídos, criando o arcabouço teórico necessário para estudos de caso acerca de PRVs específicos e suas técnicas de resolução.

Referências

[1] Ricardo Abramovay. "Funções e medidas da ruralidade no desenvolvimento contemporâneo". Em: *Instituto de Pesquisa Econômica Aplicada (IPEA)* (2000). URL: https://repositorio.ipea.gov.br/bitstream/11058/2360/1/TD_702.pdf.

[2] NR Achuthan, L Caccetta e SP Hill. "A new subtour elimination constraint for the vehicle routing problem". Em: *European Journal of Operational Research* 91.3 (1996), pp. 573–586.

[3] Pedro Amorim, Sophie N Parragh, Fabrício Sperandio e Bernardo Almada-Lobo. "A rich vehicle routing problem dealing with perishable food: a case study". Em: *Top* 22 (2014), pp. 489–508.

[4] Susana Baptista, Rui Carvalho Oliveira e Eduardo Zúquete. "A period vehicle routing case study". Em: *European Journal of Operational Research* 139.2 (2002). EURO XVI: O.R. for Innovation and Quality of Life, pp. 220–229. DOI: https://doi.org/10.1016/S0377-2217(01)00363-0. URL: https://www.sciencedirect.com/science/article/pii/S0377221701003630.

[5] Mokhtar S. Bazaraa, John J. Jarvis e Hanif D. Sherali. *Linear Programming and Network Flows*. 4ª ed. Hoboken, New Jersey: Wiley, 2010.

[6] Ramin Bazrafshan, Sarfaraz Zolfani e S.M.J. Mirzapour Al-e-hashem. "Comparison of the Sub-Tour Elimination Methods for the Asymmetric Traveling Salesman Problem Applying the SECA Method". Em: *Axioms* 10 (2021), p. 19. DOI: 10.3390/axioms10010019.

[7] John E. Beasley. *LP Relaxation*. Acessado em 18 jul 2023. URL: http://people.brunel.ac.uk/~mastjjb/jeb/or/lprelax.html.

[8] Peter Békési. *A adult pet Black-headed Caique walking on a wooden floor*. Acessado em 29 jul 2023. 2009. URL: https://commons.wikimedia.org/wiki/File:Pionites_melanocephalus_-pet_walking_on_wooden_floor-8a.jpg.

[9] Dimitris Bertsimas e John N. Tsitsiklis. *Introduction to Linear Optimization*. 4ª ed. Belmont, Massachusetts: Athena Scientific e Dynamic Ideas, 1997.

[10] Jeff Bezanson, Alan Edelman, Stefan Karpinski e Viral B. Shah. "Julia: A Fresh Approach to Numerical Computing". Em: *SIAM Review* 59.1 (2017), pp. 65–98.

[11] Tatiane Borchers e Victor Garcia Figueirôa-Ferreira. "Neoliberalismo e o esvaziamento do Estado no transporte público de Araraquara–SP". Em: *Cadernos Metrópole* 24 (2022), pp. 549–576.

[12] Stephen P. Bradley, Arnoldo C. Hax e Thomas L. Magnanti. *Applied mathematical programming*. Addison-Wesley, 1977.

[13] Raj Bridgelall. "Tutorial and Practice in Linear Programming: Optimization Problems in Supply Chain and Transport Logistics". Em: *arXiv preprint arXiv:2211.07345* (2022).

[14] Marcel Bursztyn e Flávio Eiró. "Mudanças climáticas e distribuição social da percepção de risco no Brasil". Em: *Sociedade e Estado* 30.2 (2015), pp. 471–493. URL: https://www.scielo.br/j/se/a/dTKndwcc5BcLbmb7fqyYPqr/?format=pdf&lang=pt.

[15] Andrew Butterfield, Gerard Ekembe Ngondi e Anne Kerr. *A dictionary of computer science*. Oxford University Press, 2016.

[16] Pierre Carbonnelle. *PYPL PopularitY of Programming Language*. Acessado em 20 jul 2023. URL: https://pypl.github.io/PYPL.html.

[17] Eduardo Gontijo Carrano. *Branch and Cut*. Acessado em 19 jul 2023. Universidade Federal de Minas Gerais, 2019. URL: http://www.cpdee.ufmg.br/~lusoba/disciplinas/eee933/slides/carrano/07b_BaC.pdf.

[18] Carlos Henrique Ribeiro de Carvalho. *Desafios da mobilidade urbana no Brasil*. Rel. técn. Instituto de Pesquisa Econômica Aplicada (IPEA), 2016. URL: https://repositorio.ipea.gov.br/bitstream/11058/6664/1/td_2198.pdf.

[19] Cristian Cataldo-Díaz, Rodrigo Linfati e John Willmer Escobar. "Mathematical model for the electric vehicle routing problem considering the state of charge of the batteries". Em: *Sustainability* 14.3 (2022), p. 1645.

[20] Pedro Henrique Centenaro. *Caique.jl*. https://github.com/phcentenaro7/Caique.jl. Versão 0.0.0. 2023.

[21] Mike Chalawsky, Nathan Kendrick, Daniel Lane e James Ramos. *Branch and price*. Acessado em 20 jul 2023. Cornell University, 2022. URL: https://optimization.cbe.cornell.edu/index.php?title=Branch_and_price.

[22] Jens Clausen. "Branch and bound algorithms-principles and examples". Em: *Department of Computer Science, University of Copenhagen* (1999), pp. 1–30.

[23] Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh e Daniele Vigo. "Short-Haul Routing". Em: (2005).

[24] IBM ILOG Cplex. "V12. 1: User's Manual for CPLEX". Em: *International Business Machines Corporation* 46.53 (2009), p. 157.

[25] George Dantzig, Ray Fulkerson e Selmer Johnson. “Solution of a large-scale traveling-salesman problem”. Em: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

[26] George B Dantzig e John H Ramser. “The truck dispatching problem”. Em: *Management science* 6.1 (1959), pp. 80–91.

[27] George B. Dantzig. “Origins of the Simplex Method”. Em: *A History of Scientific Computing*. New York, NY, USA: Association for Computing Machinery, 1990, pp. 141–151. URL: https://doi.org/10.1145/87252.88081.

[28] Elizabeth D. Dolan e Jorge J. Moré. “Benchmarking optimization software with performance profiles”. Em: *Mathematical Programming* 91.2 (2002), pp. 201–213. DOI: 10.1007/s101070100263. URL: https://doi.org/10.1007/s101070100263.

[29] Iain Dunning, Joey Huchette e Miles Lubin. “JuMP: A Modeling Language for Mathematical Optimization”. Em: *SIAM Review* 59.2 (2017), pp. 295–320.

[30] *Electric buses in cities report: Driving Towards Cleaner Air and Lower CO* . Acessado em 6 maio 2023. 2018. URL: https://data.bloomberglp.com/bnef/sites/14/2018/05/Electric-Buses-in-Cities-Report-BNEF-C40-Citi.pdf.

[31] Paulo Feofiloff, Yoshiharu Kohayakawa e Yoshiko Wakabayashi. “Uma introdução sucinta à teoria dos grafos”. Em: (2011).

[32] Michael C Ferris, Olvi L Mangasarian e Stephen J Wright. *Linear programming with MATLAB*. SIAM, 2007.

[33] Aurélien Froger, Jorge E Mendoza, Gilbert Laporte e Ola Jabali. “New formulations for the electric vehicle routing problem with nonlinear charging functions”. Tese de dout. Centre interuniversitaire de recherche sur les reseaux d’entreprise, la …, 2017.

[34] Armin Fügenschuh. “Solving a school bus scheduling problem with integer programming”. Em: *European Journal of Operational Research* 193.3 (2009), pp. 867–884. DOI: https://doi.org/10.1016/j.ejor.2007.10.055. URL: https://www.sciencedirect.com/science/article/pii/S0377221707010910.

[35] Armin Fügenschuh, Alexander Martin e Peter Stöveken. “Integrated optimization of school starting times and public bus services”. Em: *Operations Research Proceedings 2004: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR). Jointly Organized with the Netherlands Society for Operations Research (NGB) Tilburg, September 1–3, 2004*. Springer. 2004, pp. 150–157.

[36] Bezalel Gavish e Stephen Graves. “The Traveling Salesman Problem and Related Problems”. Em: (1978).

[37] Google. *Vehicle Routing*. https://developers.google.com/optimization/routing. Acessado em 16 set. 2022. 2021.

[38] Google. *Vehicle Routing Problem*. https://developers.google.com/optimization/routing/vrp. Acessado em 8 maio 2023. 2021.

[39] Rongge Guo, Wei Guan, Wenyi Zhang, Fanting Meng e Zixian Zhang. “Customized bus routing problem with time window restrictions: model and case study”. Em: *Transportmetrica A: Transport Science* 15.2 (2019), pp. 1804–1824.

[40] Gurobi Optimization, LLC. *Mixed-Integer Programming (MIP) – A Primer on the Basics*. https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/. Acessado em 18 jul 2023. 2022.

[41] Johan Hellmark. “A practical framework for the electric vehicle routing problem”. Em: (2022).

[42] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke e Richard F Hartl. “The electric fleet size and mix vehicle routing problem with time windows and recharging stations”. Em: *European Journal of Operational Research* 252.3 (2016), pp. 995–1018.

[43] Sérgio Buarque de Holanda, Cândido Antônio e Evaldo Cabral de Mello. *Raízes do Brasil*. 27ª ed. Companhia das Letras, 2020.

[44] Mona Issabakhsh, Seyyed-Mahdi Hosseini-Motlagh, Mir-Saman Pishvaee e Mojtaba Saghafi Nia. “A vehicle routing problem for modeling home healthcare: a case study”. Em: *International Journal of Transportation Engineering* 5.3 (2018), pp. 211–228.

[45] Brian Kallehauge, Jesper Larsen, Oli B.G. Madsen e Marius M. Solomon. “Vehicle Routing Problem with Time Windows”. Em: *Column Generation*. Ed. por Guy Desaulniers, Jacques Desrosiers e Marius M. Solomon. Boston, MA: Springer US, 2005, pp. 67–98. DOI: 10.1007/0-387-25486-2_3. URL: https://doi.org/10.1007/0-387-25486-2_3.

[46] Vaibhav Karve. *Linear and Mixed Integer Programming*. Acessado em 18 jul 2023. URL: https://vaibhavkarve.github.io/linear_programming.html.

[47] Merve Keskin e Bülent Çatay. “A matheuristic method for the electric vehicle routing problem with time windows and fast chargers”. Em: *Computers & operations research* 100 (2018), pp. 172–188.

[48] Gilbert Laporte e Yves Nobert. “Exact algorithms for the vehicle routing problem”. Em: *North-Holland mathematics studies*. Vol. 132. Elsevier, 1987, pp. 147–184.

[49] Mikhail Lavrov. *Lecture 33: Integer Programming*. Acessado em 18 jul 2023. University of Illinois Urbana-Champaign, 2019. URL: https://faculty.math.illinois.edu/~mlavrov/docs/482-fall-2019/lecture33.pdf.

[50] Zachary Leung. *Gomory Cuts and a little more*. Acessado em 20 jul 2023. Massachusetts Institute of Technology, 2012. URL: https://ocw.mit.edu/courses/15-053-optimization-methods-in-management-science-spring-2013/resources/mit15_053s13_tut11/.

[51] Catherine Lewis. *Linear Programming: Theory and Applications*. Whitman College Mathematic Department. 2008, pp. 7–9.

[52] Todd Litman. *Evaluating public transportation health benefits*. Victoria Transport Policy Institute Victoria, BC, Canada, 2012.

[53] Andrew Makhorin. *GNU Linear Programming Kit*. Department for Applied Informatics, Moscow Aviation Institute. 2013.

[54] Danijel Marković, Goran Petrović, Žarko Ćojbašić e Aleksandar Stanković. “The vehicle routing problem with stochastic demands in an urban area–a case study”. Em: *Facta Universitatis, Series: Mechanical Engineering* 18.1 (2020), pp. 107–120.

[55] C. E. Miller, A. W. Tucker e R. A. Zemlin. “Integer Programming Formulation of Traveling Salesman Problems”. Em: 7.4 (1960). DOI: 10.1145/321043.321046. URL: https://doi.org/10.1145/321043.321046.

[56] Ouaguenouni Mohamed. *A comprehensive study of Mixed Integer Programming with JuMP on Julia (Part 1) — towardsdatascience.com*. https://towardsdatascience.com/a-comprehensive-study-of-mixed-integer-programming-with-jump-on-julia-part-1-8d47418324d4. Acessado em 18 jul 2023. 2021.

[57] Fórum Econômico Mundial. *Deep Shift – Technology Tipping Points and Societal Impact*. Acessado em 1 maio 2023. 2015.

[58] Vanessa Gapriotti Nadalin e Danilo Camargo Igliori. “Evolução urbana e espraiamento na Região Metropolitana de São Paulo”. Em: *Instituto de Pesquisa Econômica Aplicada (IPEA)* (2010).

[59] Lutz Prechelt. “An empirical comparison of seven programming languages”. Em: *Computer* 33.10 (2000), pp. 23–29.

[60] Edson Prestes. “Introdução à Teoria dos Grafos”. Em: *Universidade Federal do Rio Grande do Sul, Instituto de Informática, Departamento de Informática Teórica, Tech. Rep* (2020).

[61] Santiago Valdés Ravelo. *Cutting planes and valid inequalities*. Acessado em 20 jul 2023. Universidade Federal do Rio Grande do Sul, 2020. URL: https://www.inf.ufrgs.br/~svravelo/assets/slides/PLI_11.pdf.

[62] Ivan Luiz Marques Ricarte. *Bytecodes*. Acessado em 20 jul 2023. URL: https://www.dca.fee.unicamp.br/cursos/PooJava/javaenv/bytecode.html.

[63] Amanur Rahman Saiyed. “The traveling salesman problem”. Em: *Indiana State University* 2 (2012), pp. 1–15.

[64] Luiz Rafael dos Santos. “Escolha Otimizada de Parâmetros em Métodos de Pontos Interiores para Programação Linear”. Tese de dout. Universidade Estadual de Campinas, 2014, p. 7. URL: http://repositorio.unicamp.br/Busca/Download?codigoArquivo=502752.

[65] Oliver Schade, Gregor Scheithauer e Stefan Scheler. *99 Bottles of Beer*. Acessado em 20 jul 2023. 2023. URL: http://www.99-bottles-of-beer.net/team.html.

[66] Daniel Schermer. *Traveling Salesperson Problem · JuMP*. https://jump.dev/JuMP.jl/stable/tutorials/algorithms/tsp_lazy_constraints/. Acessado em 14 maio 2023. 2023.

[67] Klaus Schwab. *A Quarta Revolução industrial*. 1ª ed. Edipro, 2019.

[68] Nasser A El-Sherbeny. “Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods”. Em: *Journal of King Saud University-Science* 22.3 (2010), pp. 123–131.

[69] Abel S. Siqueira. *Modelagem em Otimização - Caixeiro Viajante*. https://youtu.be/92H0CqHZIvc. Acessado em 16 set. 2022. 2021.

[70] SOBRAPO. *O que é pesquisa operacional?* 2017. URL: https://www.sobrapo.org.br/o-que-e-pesquisa-operacional#:~:text=Pesquisa%20Operacional%20(PO)%20%C3%A9%20a,diversas%20%C3%A1reas%20de%20atua%C3%A7%C3%A3o%20humana..

[71] Gregório Costa Luz de Souza Lima, Gabriel Lassery Rocha da Silva e Genezio dos Santos Albuquerque Neto. “Mobilidade elétrica: o ônibus elétrico aplicado ao transporte público no Brasil”. Em: *Revista dos Transportes Públicos-ANTP-Ano* 41 (2019), 2º.

[72] KR Srinath. “Python–the fastest growing programming language”. Em: *International Research Journal of Engineering and Technology* 4.12 (2017), pp. 354–357.

[73] Eliana M Toro O, Antonio H Escobar Z e Mauricio Granada E. “Literature review on the vehicle routing problem in the green transportation context”. Em: *Luna Azul* 42 (2016), pp. 362–387.

[74] Linus Torvalds. *Linux kernel source tree*. Acessado em 20 jul 2023. 2023. URL: https://github.com/torvalds/linux.

[75] REAMAT - UFRGS. *Fatoração LU*. https://www.ufrgs.br/reamat/CalculoNumerico/livro-oct/sdsl-fatoracao_lu.html. Acessado em 22 abr. 2023. 2020.

[76] Heloisa Passarelli Vieira. “Metaheuristica para a Solução de Problemas de Roteamento de Veículos com Janela de Tempo”. Diss. de mestr. Universidade Estadual de Campinas, 2013. URL: https://www.ime.unicamp.br/~chico/tese_heloisa.pdf.

[77] Ryan Waite. *What Programming Language is Windows written in?* Acessado em 20 jul 2023. 2009. URL: https://social.microsoft.com/Forums/en-US/65a1fe05-9c1d-48bf-bd40-148e6b3da9f1/what-programming-language-is-windows-written-in?forum=windowshpcacademic.

[78] Leonardo Zambito. “The traveling salesman problem: a comprehensive survey”. Em: *Project for CSE* 4080 (2006).

[79] Le Zhang, Shuaian Wang e Xiaobo Qu. “Optimal electric bus fleet scheduling considering battery degradation and non-linear charging profile”. Em: *Transportation Research Part E: Logistics and Transportation Review* 154 (2021), p. 102445.

[80] Xiaorong Zuo, Yiyong Xiao, Meng You, Ikou Kaku e Yuchun Xu. “A new formulation of the electric vehicle routing problem with time windows considering concave nonlinear charging function”. Em: *Journal of Cleaner Production* 236 (2019), p. 117687.

A. O método simplex

A.1. Teoremas fundamentais para o método simplex

Definição A.1 (Reta)**.** Um poliedro $\mathbf{P} \subset \mathbb{R}^n$ contém uma *reta* se existem um vetor $x \in \mathbf{P}$ e um vetor não-nulo $d \in \mathbb{R}^n$ tais que $x + \lambda d \in \mathbf{P}$ para todo escalar λ.

Teorema A.2. *Suponha que o poliedro* $\mathbf{P} = \{x \in \mathbb{R}^n \mid a_i^\intercal x \leq b_i, i = 1, \ldots, m\}$ *é não-vazio. Então segue-se que:*

(a) $\mathbf{P}$ *tem pelo menos um ponto extremo.*

(b) $\mathbf{P}$ *não contém reta.*

(c) Existem n *vetores da família* $a_1, \ldots, a_m$ *que são linearmente independentes.*

Demonstração. $(b) \Rightarrow (a)$

Sejam $x \in \mathbf{P}$ e $\mathbf{I} = \{i \mid a_i^\intercal x = b_i\}$. Se n vetores $a_i, i \in \mathbf{I}$, forem linearmente independentes, então x é, por definição, uma solução básica factível e, portanto, existe ponto extremo em $\mathbf{P}$. Se este não for o caso, todos os vetores $a_i, i \in \mathbf{I}$, pertencem a um subespaço próprio de $\mathbb{R}^n$ e, destarte, existe um vetor $d \neq \mathbf{0}$ tal que $a_i^\intercal d = 0$ para todo $i \in \mathbf{I}$. Seja $y = x + \lambda d$ uma reta, com λ sendo um escalar arbitrário. Para $i \in \mathbf{I}$, temos

$$a_i^\intercal y = a_i^\intercal x + \lambda a_i^\intercal d = a_i^\intercal x = b_i.$$

Logo, as restrições ativas em x permanecem ativas ao longo de toda a reta y. Contudo, como estamos assumindo (b) como hipótese, segue-se que, variando λ, estaremos eventualmente violando uma restrição. Na iminência de uma tal violação, uma nova restrição deve ficar ativa. Logo, existem λ^* e $j \notin \mathbf{I}$ tais que $a_j^\intercal(x + \lambda^* d) = b_j$.

Vamos provar que a_j não é combinação linear dos vetores $a_i, i \in \mathbf{I}$. Temos que $a_j^\intercal x \neq b_j$ (pois $j \notin \mathbf{I}$) e $a_j^\intercal(x + \lambda^* d) = b_j$ (pela definição de λ^*). Assim, $a_j^\intercal d \neq 0$. Por outro lado, $a_i^\intercal d = 0$ para todo $i \in \mathbf{I}$ (pela definição de d) e, portanto, d é ortogonal a qualquer combinação linear dos vetores $a_i, i \in \mathbf{I}$. Como d não é ortogonal a a_j, concluímos que a_j não é combinação linear dos vetores $a_i, i \in \mathbf{I}$.

Logo, no deslocamento do ponto x para $x + \lambda^* d$, o número de restrições linearmente independentes aumenta em pelo menos um. Repetindo este argumento tantas vezes quanto for necessário, eventualmente chegamos a um ponto em que n restrições são ativas. Por definição, este ponto é solução básica e, como não violamos nenhuma restrição, é solução básica factível e, portanto, um ponto extremo.

$(a) \Rightarrow (c)$

Se $x \in \mathbf{P}$ é ponto extremo, x também é solução básica factível, e existem n restrições ativas em x cujos vetores a_i são linearmente independentes.

$(c) \Rightarrow (b)$

Suponha que n dos vetores a_i sejam linearmente independentes e, sem perda de generalidade, sejam $a_1, \ldots, a_n$ linearmente independentes. Suponha que $\mathbf{P}$ contenha uma reta $x+\lambda d$, com $d \neq \mathbf{0}$. Então, temos que $a_i^\intercal(x+\lambda d) \geq b_i$ para todo i e todo λ. Concluímos que $a_i^\intercal d = 0$ para todo i. Como os vetores $a_i, i = 1, \ldots, n$ são linearmente independentes, segue-se que $d = \mathbf{0}$. Mas isso contradiz o que acabamos de definir para d. Portanto, $\mathbf{P}$ não contém retas. □

Finalmente, estamos em condições de introduzir o teorema que justifica a ideia central do método simplex.

Teorema A.3. *Considere o problema de otimização linear de minimizar $c^\intercal x$ em um poliedro $\mathbf{P}$. Suponha que $\mathbf{P}$ tenha pelo menos um ponto extremo. Então, ou o custo ótimo é igual a $-\infty$, ou existe um ponto extremo ótimo.*

Demonstração. Para esta demonstração, considere a seguinte terminologia: um elemento $x \in \mathbf{P}$ tem *posto* k se for possível encontrar k, e não mais que k, restrições linearmente independentes ativas em x.

Suponha que o valor objetivo ótimo é finito. Seja $\mathbf{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ e considere $x \in \mathbf{P}$ com posto $k < n$. Vamos provar a existência de um $y \in \mathbf{P}$ com posto maior que k que também satisfaça $c^\intercal y \leq c^\intercal x$. Seja $\mathbf{I} = \{i \mid a_i^\intercal x = b_i\}$, onde $a_i^\intercal$ é a i-ésima linha de A. Como $k < n$, os vetores $a_i, i \in \mathbf{I}$, pertencem a um subespaço próprio de $\mathbb{R}^n$, e podemos escolher um vetor $d \in \mathbb{R}^n, d \neq \mathbf{0}$, ortogonal a todo $a_i, i \in \mathbf{I}$. Ademais, considere que d é tal que $c^\intercal d \leq 0$.

Suponha que $c^\intercal d < 0$ e considere a semirreta $y = x + \lambda d$, onde λ é um escalar positivo. Como se segue do Teorema A.2, todos os pontos nesta semirreta satisfazem as relações $a_i^\intercal y = b_i, i \in \mathbf{I}$. Se toda a semirreta estivesse contida em $\mathbf{P}$, o valor objetivo ótimo seria $-\infty$. Como esta não é nossa hipótese, concluímos que a semirreta sai de $\mathbf{P}$ a partir de algum ponto. Na iminência desta saída, temos $\lambda^* > 0$ e $j \notin \mathbf{I}$ tais que $a_j^\intercal(x+\lambda^* d) = b_j$. Então, sendo $y = x + \lambda^* d$, vale que $c^\intercal y < c^\intercal x$. Consequentemente, como mostrado no teorema mencionado, a_j é linearmente independente em relação a $a_i, i \in \mathbf{I}$, e o posto de y é pelo menos $k + 1$.

Agora suponha que $c^\intercal d = 0$. Seja $y = x + \lambda d$ uma reta com λ escalar arbitrário. Como $\mathbf{P}$ não contém retas, a reta precisa sair de $\mathbf{P}$ em algum ponto e, na iminência disso, temos outro vetor y de posto maior que x. Ademais, como $c^\intercal d = 0$, temos $c^\intercal y = c^\intercal x$.

Em ambos os casos, encontramos y tal que $c^\intercal y \leq c^\intercal x$ e $\text{posto}(y) > \text{posto}(x)$. Repetindo este processo quantas vezes for necessário, chegamos a um vetor w tal que $\text{posto}(w) = n$ e $c^\intercal w \leq c^\intercal x$.

Sejam $w^1, \ldots, w^r$ as soluções básicas factíveis em $\mathbf{P}$ e seja w^* uma solução básica factível tal que $c^\intercal w^* \leq c^\intercal w^i$ para todo i. Já demonstramos que para todo x existe i tal que $c^\intercal w^i \leq c^\intercal x$. Segue-se que $c^\intercal w^* \leq c^\intercal x$ para todo $x \in \mathbf{P}$, e a solução básica factível w^* é ótima. □

Note que se um poliedro não-vazio não tiver ponto extremo, sua solução ótima é $-\infty$. Pelo contrário, se o poliedro for não-vazio e tiver ponto extremo, sua solução ótima pode ser $-\infty$ ou está localizada em algum dos pontos extremos. Isso nos permite chegar à conclusão necessária para começar a desenvolver o método simplex.

Corolário A.4. *Considere o problema de otimização linear de minimizar $c^\intercal x$ em um poliedro não-vazio. Então o custo ótimo é $-\infty$ ou existe solução ótima.*

Deste ponto em diante, chamaremos o caso em que a solução ótima do problema é $-\infty$ de *caso ilimitado*.

A.2. Considerações iniciais

Agora que sabemos o exposto pelo Corolário A.4, podemos pensar em uma forma rudimentar de resolver problemas de otimização linear. Uma forma conceitualmente simples é descrita nos passos a seguir.

1. Encontrar todas as soluções básicas $x^1, \ldots, x^p$.
2. Determinar o conjunto $\mathbf{F} = \{x^i \mid x^i \text{ é factível}\}$.
3. Encontrar o ponto ótimo $x^* = \min(\mathbf{F})$.

Evidentemente, esta abordagem tem problemas. Primeiramente, esta verificação não considera a possibilidade de a solução ótima ser $-\infty$. Além disso, como evidenciado pela Equação (30), existem $\binom{n}{m}$ maneiras de organizar as n variáveis de decisão do problema de maneira que o sistema $Ax = b$ seja satisfeito junto com as restrições de não-negatividade; ou, ainda, existem $p = \binom{n}{m}$ soluções básicas para o problema. Logo, para problemas com número considerável de variáveis e restrições, este método é custoso já no primeiro passo, tornando-se praticamente inviável.

O método simplex implementa uma ideia mais sofisticada do exposto acima. Este método depende de um ponto extremo inicial e, a partir deste ponto, se desloca para um novo ponto extremo a cada iteração. O deslocamento do simplex é condicionado de modo que o método reduza o valor objetivo a cada iteração. Além disso, o método é capaz de determinar se o ponto em que se encontra é ótimo, ou se a solução ótima é $-\infty$, sem ter que compará-lo a todos os outros pontos extremos.

Implementamos o método simplex através de um pacote computacional chamado `Caique.jl` [20], que se encontra em um repositório do *GitHub* (`https://github.com/phcentenaro7/Caique.jl`). O `Caique.jl` é um pacote escrito em Julia [10] que pode ser usado para resolver problemas de otimização linear. O código do pacote foi implementado com base nas obras de Bazaraa, Jarvis e Sherali [5] e Bertsimas e Tsitsiklis [9], e põe em prática os algoritmos que serão apresentados nesta seção.

Caique é o nome em inglês de uma espécie de pássaros conhecida no Brasil como marianinha. Como o método simplex "pula" de um ponto extremo do problema a outro, ele lembra bastante uma das muitas marianinhas saltitantes que podem ser encontradas em vídeos pela internet, servindo de inspiração para o nome do pacote.

Figura 12: Exemplo de marianinha [8]

A.3. Representação de um problema em forma canônica

Antes de começar o desenvolvimento do simplex, precisamos garantir que o problema de otimização linear esteja na forma *canônica*. Diferentemente da forma padrão, apresen-

tada no Problema (24), a forma canônica se apresenta em termos apenas de igualdades, à exceção apenas das restrições de não-negatividade, que continuam as mesmas. O Problema (75) apresenta a forma canônica.

$$\min\ c^\intercal x \tag{75}$$
$$\text{s.a } Ax = b, \tag{76}$$
$$x \geq \mathbf{0}. \tag{77}$$

Neste caso, assim como na forma padrão, A é uma matriz de dimensões $m \times n$, c e x são vetores no $\mathbb{R}^n$ e b é um vetor no $\mathbb{R}^m$.

É possível converter problemas da forma padrão para a forma canônica de modo que continuem os mesmos. Para tal, faremos uso de *variáveis de folga*.

Definição A.5 (Variável de folga)**.** Dada uma restrição de desigualdade de um problema de otimização linear, uma nova variável x_s introduzida à restrição é definida como *variável de folga* se:

- $a_i^\intercal x + x_s = b_i$, para $a_i^\intercal x \leq b_i$.
- $a_i^\intercal x - x_s = b_i$, para $a_i^\intercal x \geq b_i$.

Note que, se adicionarmos uma variável de folga x_s à i-ésima restrição do problema, o vetor dos coeficientes de x_s para cada restrição será o i-ésimo vetor da base canônica, ou e_i. Por outro lado, se subtrairmos x_s da restrição, então o vetor de coeficientes de x_s será $-e_i$.

Usando variáveis de folga, podemos converter qualquer restrição de desigualdade em restrição de igualdade. Note que este processo gera um problema equivalente ao da forma padrão. A única diferença é que a variável representante da quantidade que falta para igualar os dois lados passa a ser explicitada.

Por razões que veremos no Apêndice A.8, é importante garantir que o vetor de valores do lado direito das equações, b, seja não-negativo. Sendo $i = 1, \ldots, m$, isso pode ser feito multiplicando por -1 as equações para as quais $b_i < 0$. Equivalentemente, podemos multiplicar as inequações ainda no formato padrão do problema por -1.

A implementação do procedimento que permite a conversão de um problema para a forma canônica é feita no `Caique.jl`, na função `createSlackSubmatrix`, nos moldes do Algoritmo 1. No código, existe a estrutura imutável `LinearProgram`, que recebe, como argumentos para sua construção, a matriz A e os vetores b, c, x e, ainda, um vetor s, que

contém valores conhecidos na linguagem como *símbolos*. Um símbolo é uma sucessão de caracteres, sem espaços, precedida por dois pontos. No caso, usamos o vetor de símbolos s para definir se as restrições que estão sendo passadas ao construtor são de igualdade, maior igual ou menor igual (`:equal`, `:greater` e `:less` na implementação, respectivamente). Para adicionar as variáveis de folga ao problema, o objetivo é criar uma matriz X^s que represente a adição ou subtração das variáveis de folga necessárias. Esta matriz pode ser concatenada à direita da matriz A, de modo que a nova matriz A passe a representar também as variáveis de folga.

Note que, a partir daqui, estaremos nos referindo à i-ésima linha de uma matriz A qualquer por A_i, e à i-ésima coluna desta matriz por a_i, por simplicidade de notação.

Algoritmo 1 Adicionar variáveis de folga ao problema (Implementação)

1: criar uma matriz X^s de dimensões $m \times 0$, que representa as restrições das variáveis de folga
2: definir $i \leftarrow 1$, que será nosso contador de linhas
3: definir $k \leftarrow 1$, que será o contador do número de variáveis de folga adicionadas ao problema
4: **enquanto** $i \leq m$ **fazer**
5: **se** $b_i < 0$ **então**
6: redefinir o valor do lado direito $b_i \leftarrow -(b_i)$
7: redefinir a linha $A_i \leftarrow -A_i$
8: inverter o sinal da restrição, se for uma inequação
9: **se** o sinal da restrição for $\leq$ **então**
10: redefinir $k \leftarrow k + 1$
11: redefinir $x_S \leftarrow [x_S, e_i]$
12: **senão, se** o sinal da restrição for $\geq$ **então**
13: redefinir $k \leftarrow k + 1$
14: redefinir $x_S \leftarrow [x_S, -e_i]$
15: redefinir $i \leftarrow i + 1$
16: Concatenar k zeros ao vetor de custos, um para cada variável de folga adicionada ao problema
17: **retornar** a matriz X^s atual, pois chegamos ao fim do algoritmo

Após o término deste algoritmo, a matriz X^s pode ser concatenada à direita da matriz A, produzindo a representação em forma canônica do problema, como desejado.

A.4. Variáveis básicas e não-básicas

Como explicado na introdução desta seção, para que o método simplex funcione, precisamos começar o problema em uma solução básica factível. Note, pela Definição 3.29,

que, para que um ponto seja solução básica factível, ele precisa, entre outras coisas, ter n restrições linearmente independentes ativas nele. Neste trabalho, vamos considerar um problema de otimização linear na forma canônica com posto$(A) = m$. Então, segue-se que $n \geq m$. O que nos interessa é o caso não-trivial em que $n > m$. Note que podemos encontrar uma solução básica seguindo os passos a seguir.

1. Zerar $n - m$ das variáveis de decisão do problema.

2. Resolver o sistema de equações $Ax = b$.

Note que, ao fixar $n - m$ variáveis de decisão em zero, sobram m variáveis para resolver um sistema de m equações. Ou seja, os dois passos acima nos permitem trivialmente encontrar soluções básicas, bastando permutar as variáveis que definimos como zero. Note, ainda, que isso não nos garante que a solução básica encontrada seja *factível*; ou seja, pode ser que, com as variáveis que escolhemos zerar, o sistema de equações $Ax = b$ não resulte em $x \geq \mathbf{0}$.

Para problemas pequenos, é possível permutar soluções básicas até que se verifique que uma delas é factível; no entanto, como mostrado pela Equação (30), este procedimento é computacionalmente inviável para problemas grandes. Como veremos adiante, é possível definir um novo problema de otimização linear a partir do original que, fornecido ao método simplex, pode nos retornar uma solução básica factível para o problema original. Todavia, isso requer que primeiro desenvolvamos o método simplex. Então vamos supor, por ora, que sempre temos uma solução básica factível pela qual começar.

Agora que temos um método para obter soluções básicas, é útil definir termos muito importantes para desenvolvimento do método simplex. Sendo x o vetor de variáveis de decisão do problema de otimização linear, valem as seguintes definições.

Definição A.6 (Variável não-básica)**.** Uma variável $x_i, i \in \{1, \ldots, m\}$, é dita *não-básica* quando $x_i = 0$ para satisfazer uma das restrições de não-negatividade do problema.

Definição A.7 (Variável básica)**.** Uma variável $x_i, i \in \{1, \ldots, m\}$, é dita *básica* quando ela constitui solução do problema $Ax = b$ após a seleção das variáveis não-básicas.

Definição A.8 (Matriz base)**.** Assuma, sem perda de generalidade, que as variáveis básicas do problema tenham índices $1, \ldots, m$. Então a *matriz base* do problema é definida como $B = [a_1, \ldots, a_m]$.

Definição A.9 (Matriz não-base)**.** Assuma, sem perda de generalidade, que as variáveis não-básicas do problema tenham índices $m + 1, \ldots, n$. Então a *matriz não-base* do problema é definida como $N = [a_{m+1}, \ldots, a_n]$.

Feitas estas definições, podemos reescrever o Problema (75) como

$$\min\ c_B^\intercal x_B + c_N^\intercal x_N \tag{78}$$
$$\text{s. a } Bx_B + Nx_N = b, \tag{79}$$
$$x_B, x_N \geq \mathbf{0}, \tag{80}$$

onde B é a matriz base, N é a matriz não-base, x_B é o vetor de variáveis básicas e x_N é o vetor de variáveis não-básicas.

Por fim, introduzimos o conceito de degeneração em poliedros, que é uma extensão da Definição 3.31.

Definição A.10 (Degeneração em poliedros)**.** Considere o poliedro na forma canônica $\mathbf{P} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq \mathbf{0}\}$, sendo x solução básica. Seja A uma matriz $m \times n$. O vetor x é uma solução básica *degenerada* se o número de componentes nulos de x for maior do que $n - m$.

Da maneira como estamos construindo soluções básicas, só é possível uma destas soluções ser degenerada se uma das variáveis básicas tiver que ser igual a zero para resolver $Ax = b$. Neste caso, além das n restrições ativas esperadas, surgem k restrições de não-negatividade ativas a mais do que o esperado, para k variáveis básicas iguais a zero. Assim, a solução básica em questão será ativa em $n + k$ restrições. Como estamos no $\mathbb{R}^n$, isso significa que k restrições ativas são linearmente dependentes em relação às demais.

Como veremos no Apêndice A.9, pontos degenerados podem travar o método simplex no lugar, requerendo métodos especiais para superá-los.

A.5. Mecanismo iterativo de redução do valor objetivo

Nosso objetivo agora é verificar como o simplex garante que o valor objetivo é reduzido de uma iteração para a outra. Podemos continuar decompondo o Problema (78) para chegar aonde queremos.

Começamos pré-multiplicando ambos os lados do sistema de equações pela inversa da base, obtendo

$$x_B + B^{-1}Nx_N = B^{-1}b. \tag{81}$$

Denotando por $j_{(1)}, \ldots, j_{(n-m)}$ os índices das variáveis não-básicas, note que

$$Nx_N = \begin{bmatrix} a_{j(1)} & \dots & a_{j(n-m)} \end{bmatrix} \begin{bmatrix} x_{j(1)} \\ \vdots \\ x_{j(n-m)} \end{bmatrix} \tag{82}$$

$$= (a_{j(1)}x_{j(1)} + \dots + a_{j(n-m)}x_{j(n-m)}). \tag{83}$$

Ou seja, sendo **J** o conjunto de índices das variáveis não-básicas do problema, podemos escrever

$$B^{-1}Nx_N = B^{-1}\sum_{j\in J} a_j x_j = \sum_{j\in J} B^{-1}a_j x_j. \tag{84}$$

Agora vamos definir $\bar{b} = B^{-1}b$ e reescrever a equação em termos do conjunto **J**:

$$x_B + \sum_{j\in \mathbf{J}} B^{-1}a_j x_j = \bar{b}. \tag{85}$$

Seja $y_j = B^{-1}a_j, j \in \mathbf{J}$. Então, substituindo os termos na equação e isolando x_B, temos:

$$x_B = \bar{b} - \sum_{j\in \mathbf{J}} y_j x_j. \tag{86}$$

Note que isso nos permite reescrever a expressão da função objetivo do Problema (78) como

$$z = c_B^\intercal \left(\bar{b} - \sum_{j\in \mathbf{J}} y_j x_j \right) + \sum_{j\in \mathbf{J}} c_j x_j, \tag{87}$$

onde c_B é o vetor de custos das variáveis básicas.

Considere $z_0 = c_B^\intercal \bar{b}$ e $z_j = c_B^\intercal y_j$. Então ficamos com

$$z = z_0 - \sum_{j\in \mathbf{J}} z_j x_j + \sum_{j\in \mathbf{J}} c_j x_j. \tag{88}$$

Por fim, podemos unir as expressões dos somatórios como

$$z = z_0 - \sum_{j\in \mathbf{J}} (z_j - c_j) x_j, \tag{89}$$

e podemos simplificar a expressão considerando $\bar{z_j} = z_j - c_j$:

$$z = z_0 - \sum_{j \in \mathbf{J}} \bar{z}_j x_j. \tag{90}$$

O Problema (91) mostra o resultado da nova representação do Problema (78).

$$\min\ z_0 - \sum_{j \in \mathbf{J}} \bar{z}_j x_j \tag{91}$$

$$\text{s. a } x_B + \sum_{j \in \mathbf{J}} y_j x_j = \bar{b}, \tag{92}$$

$$x_B, x_N \geq \mathbf{0}. \tag{93}$$

Perceba que reescrevemos o problema de modo que as variáveis básicas passaram a cumprir papel de variáveis de folga. Ou seja, nesta representação, as variáveis básicas aparecem uma única vez, em restrições diferentes. Isso é excelente para o método simplex, pois nos permite obter facilmente os valores de x_B conforme x_N varia.

Para entender este ponto, vamos imaginar que estejamos em uma solução básica factível. Então, segue-se que $x_N = \mathbf{0}$ e, portanto, $x_B = \bar{b}$. Além disso, $z = z_0$. Disso tiramos que z_0 é o valor objetivo do problema na solução atual.

Agora imagine que queiramos reduzir o valor objetivo. Evidentemente, isso pode ser feito aumentando o valor de um variável não-básica x_k, para a qual $\bar{z_k} > 0$. Se não existir k tal que $\bar{z_k}$ satisfaça esta condição, então o ponto atual é ótimo. Do contrário, geralmente é mais interessante escolher x_k para a qual $\bar{z_k}$ seja máximo, pois então a redução unitária do valor objetivo é a maior possível. Qualquer que seja a variável escolhida para ter seu valor aumentado, passamos a chamá-la de *variável de entrada*. Para entender este nome, vamos analisar o que acontece quando escolhemos aumentar o valor da variável x_k. Isolando x_B no Problema (91), temos

$$x_B = \bar{b} - y_k x_k. \tag{94}$$

Em particular, imagine uma variável básica de índice r para a qual $y_{kr} > 0$. Segue-se que, quanto mais o valor de x_k for aumentado, mais x_r diminuirá. Esta diminuição só pode ocorrer até que $x_r = 0$, pois, do contrário, x_r estará violando sua restrição de não-negatividade.

Imagine agora que x_r seja a primeira variável que fica igual a zero quando x_k aumenta, ou seja, x_r define quando o crescimento de x_k deve parar e, por isso, x_r é chamada de *variável de bloqueio*. Perceba que, quando isso acontece, x_k vira variável básica e

x_r vira variável não-básica. Por isso, a variável de bloqueio também é conhecida como *variável de saída*, e dizemos que x_k *entra* na base e x_r *sai* da base.

Feito este raciocínio, sabemos que o valor de x_k após a detecção da variável de bloqueio será

$$x_k = \frac{\bar{b_r}}{y_{kr}}. \tag{95}$$

Note que as alterações que as variáveis básicas sofrem não alteram o valor objetivo, e note também que x continua satisfazendo n restrições linearmente independentes. Portanto, o ponto para o qual nos deslocamos é vértice. Mais especificamente, como tudo que fizemos foi trocar a restrição de não-negatividade ativa, realizamos um deslocamento até uma solução básica factível adjacente.

Por fim, suponha que $y_k \leq 0$. Neste caso, note que x_k reduz o valor do lado esquerdo de todas as restrições do Problema (91) nas quais ele aparece. Assim, as variáveis básicas aumentarão nas proporções necessárias para garantir que o sistema de equações continue sendo satisfeito. Todas as demais variáveis continuam respeitando as restrições de não-negatividade e, portanto, temos a garantia de que estamos em um ponto factível, qualquer que seja o valor para o qual aumentarmos x_k. Logo, quando $x_k \to \infty$, sabemos que $z \to -\infty$, e concluímos que a semirreta

$$\begin{bmatrix} x_B \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} -y_k \\ e_k \end{bmatrix} x_k \tag{96}$$

indica um sentido de redução infinita do valor objetivo do problema. Com isso, provamos que o método simplex é capaz de determinar se um problema tem caso ilimitado.

A.6. Uma formulação inicial do simplex

Agora que conhecemos o mecanismo iterativo do simplex, estamos habilitados a descrever o método em forma de passo a passo. Na sua forma mais simples, o simplex pode ser descrito como o Método 2.

Vamos discutir os passos principais em mais detalhes a seguir.

Atualização das variáveis

Como o simplex lida com pontos extremos do problema de otimização linear, segue-se que a cada iteração precisamos, no mínimo, de dados que descrevam quais são os valores

Método 2 Simplex (Implementação)

1: definir iter $\leftarrow 1$ como o número de iterações
2: definir maxiter como o número máximo de iterações
3: **enquanto** a solução ótima for desconhecida e iter $<$ maxiter **fazer**
4: realizar a *atualização das variáveis*
5: realizar a operação de *precificação* e **retornar** o ponto ótimo, se existir
6: realizar o *teste do caso ilimitado* e **retornar** a semirreta ótima, se existir
7: realizar o *teste da razão mínima*
8: trocar as variáveis de entrada e de saída
9: redefinir iter $\leftarrow$ iter $+ 1$
10: **retornar** que o número máximo de iterações foi atingido sem encontrar uma solução

atuais das variáveis básicas, qual é a matriz base e qual é o valor objetivo do problema. Assim, o passo de atualização das variáveis pode ser formulado como no Algoritmo 3.

Algoritmo 3 Atualização das variáveis (Implementação)

1: atualizar a matriz base $B \leftarrow [a_{i(1)}, \ldots, a_{i(m)}]$, onde $i(1), \ldots, i(m)$ são os índices das variáveis básicas
2: atualizar os valores das variáveis básicas $x_B \leftarrow B^{-1}b$
3: atualizar $\bar{b} \leftarrow x_B$
4: atualizar o valor objetivo $z \leftarrow c^\intercal x_B$

A partir deste conjunto simples de dados, todos os demais passos de uma iteração do método simplex funcionam.[24]

Precificação

A precificação é o processo preliminar para a seleção de uma variável de entrada, discutida no Apêndice A.5. O Algoritmo 4 descreve este processo.

Se a precificação não retornar ponto ótimo, então temos uma variável não-básica x_k com potencial para entrar na base.

Teste de caso ilimitado

Quando a precificação não retorna ponto ótimo, devemos verificar a possibilidade de que x seja o vértice de uma semirreta que leva a solução ótima a $-\infty$. Este teste é descrito no Algoritmo 5.

[24] No `Caique.jl`, aproveitamos a *fatoração LU* [75] do Julia para otimizar operações que envolvam a matriz base. Assim, a matriz base (`B`) serve apenas como uma forma de chegar à sua fatoração LU (`Blu`), podendo ser descartada em seguida.

Algoritmo 4 Precificação (Implementação)

definir $w \leftarrow c_B^\intercal B^{-1}$, conhecido como *vetor de multiplicadores simplex*
definir a matriz não-base como $N \leftarrow [a_{j(1)}, \ldots, a_{j(m)}]$, onde $j(1), \ldots, j(m)$ são os índices das variáveis não-básicas
definir o vetor de reduções unitárias de custo $\bar{z} \leftarrow N^\intercal w - c_N$
obter a variável não-básica x_k tal que $\bar{z}_k = \max(\bar{z})$
se $\bar{z}_k \leq 0$ **então**
    retornar a solução atual, pois ela é ótima
retornar o índice k da variável de entrada

Algoritmo 5 Teste de caso ilimitado (Implementação)

definir $y_k \leftarrow B^{-1} a_k$
obter $y_{k*} \leftarrow \max(y_k)$
se $y_{k*} \leq 0$ **então**
    retornar a Semirreta (96), pois a solução atual é vértice da semirreta, que descreve a direção na qual a solução ótima tende a $-\infty$

Teste da razão mínima

No caso de os dois algoritmos anteriores não terem retornado solução ótima, concluímos que ainda é possível realizar uma otimização finita do valor objetivo. Pela Equação (95), note que

$$x_k = \min_{1 \leq i \leq m} \left\{ \frac{\bar{b}_i}{y_{ik}}, y_{ik} > 0 \right\} \tag{97}$$

deve ser o valor da variável de entrada, com o índice r da variável de saída igual ao i para o qual a Equação (97) é mínima.

Esta maneira de encontrar a variável de saída é conhecida como teste da razão mínima. O Algoritmo 6 expressa uma forma de implementá-la.[25]

Com isso, definimos todo o método simplex na sua forma mais básica. Mas esta forma tem limitações consideráveis. Por exemplo, ainda não desenvolvemos um método para obter uma solução básica factível inicial que não recorra a força bruta ou a um palpite. Nosso método também é inútil caso encontre um ponto degenerado durante a resolução de um problema.

Apesar das limitações, esta versão do simplex pode ser modificada de modo a soluci-

[25] É importante destacar que existem várias maneiras de implementar a checagem da variável de saída. Apesar de a lógica matemática indicar que devemos ignorar todos os $y_{ik} \leq 0$, a lógica de programação pode seguir um caminho diferente. No caso do `Caique.jl`, aproveitamos a alta abstração de Julia para definir todas as razões com $y_{ik} \leq 0$ como ∞ (ou `Inf`, na terminologia do Julia). Isso facilita a obtenção do índice r, mas ao custo de mais memória.

Algoritmo 6 Teste da razão mínima (Implementação)

definir um vetor vazio q de razões
definir $i \leftarrow 1$
enquanto $i \leq m$ **fazer**
    se $y_{ik} \leq 0$ **então**
        ignorar $\bar{b_i}/y_{ik}$
    senão,
        concatenar $\bar{b_i}/y_{ik}$ ao vetor q
retornar a variável de saída r tal que $\bar{b_r}/y_{rk}$ é o menor valor do vetor q

onar todos estes problemas. É isso que abordaremos depois que provarmos que o método desenvolvido converge para uma solução ótima em um número finito de iterações.

Prova de convergência finita

Teorema A.11. *Na ausência de degeneração, o método simplex termina em um número finito de iterações, com a conclusão de que existe uma solução básica factível ótima ou que o valor objetivo ótimo é* $-\infty$.

Demonstração. Em cada iteração, ocorre uma de três coisas:

(a) Para-se em uma solução básica factível ótima, quando $\bar{z_k} \leq \mathbf{0}$.

(b) Para-se no vértice de uma solução ilimitada, quando $\bar{z_k} > \mathbf{0}$ e $y_k \leq \mathbf{0}$.

(c) É obtida uma nova solução básica factível de valor objetivo reduzido, quando $\bar{z_k} > \mathbf{0}$ e $y_k \not\leq \mathbf{0}$.

Pelo item (c), concluímos que, de uma iteração para outra, o método simplex se desloca para uma solução básica factível diferente, sem repetições. Além disso, pelo Teorema 3.32, existe um número finito de soluções básicas factíveis. Logo, o método simplex encontra um ponto que satisfaça (a) ou (b) em um número finito de iterações. □

A.7. O simplex tableau

O método desenvolvido na subseção anterior é difícil de acompanhar. Perceba que só mantemos os valores relevantes para uma variável não-básica na memória – a variável de entrada. De resto, não sabemos que mudanças estão acontecendo. Não sabemos quais outras variáveis tinham potencial de entrar na base, nem de que maneira a escolha da variável de entrada afetou as demais.

Precisamos de outra maneira de representar – ou mesmo de pensar – o simplex para poder entender melhor o que acontece a cada iteração. Não à toa, para discutir conceitos mais avançados sobre o método, costuma-se apresentá-lo em formato de tableau [9].

A.7.1. Construção do tableau inicial

Existem múltiplas formas de apresentar o simplex tableau, mas seguiremos a usada por Bazaraa, Jarvis e Sherali [5]. Sem perda de generalidade, suponha que as variáveis básicas estão organizadas com os índices $1, \ldots, m$, enquanto as variáveis não-básicas estão organizadas com os índices $m+1, \ldots, n$. Então construímos o tableau do modo indicado a seguir.

	z	x_1	$\ldots$	x_m	x_{m+1}	$\ldots$	x_n	RHS
z	1	0	$\ldots$	0	$\bar{z}_{m+1}$	$\ldots$	$\bar{z_n}$	$c^\intercal \bar{b}$
x_1	0							
$\vdots$	$\vdots$		$\mathbf{I}$			$B^{-1}N$		$\bar{b}$
x_m	0							

Tabela 4

A partir daqui, vamos chamar a linha de z de linha zero e a coluna de z de coluna zero. Note que, em termos das variáveis não-básicas, a linha zero contém todos os respectivos coeficientes de redução unitária do valor objetivo. Já para as linhas $1, \ldots, m$, remonte à Equação (81). Perceba que estas linhas mostram os valores dos coeficientes das variáveis não-básicas nas restrições de igualdade da nova representação do problema. Isso também explica por que as linhas $1, \ldots, m$ das variáveis básicas formam uma matriz identidade – afinal, como discutido logo após a apresentação do Problema (91), as variáveis básicas atuam como variáveis de folga (ver discussão sobre variáveis de folga no Apêndice A.3).

Quanto à linha zero das variáveis básicas, chegamos à conclusão de que é preenchida por zeros através de uma conta simples. Suponha uma variável básica $x_i, i \in \{1, \ldots, m\}$. Então a expressão da sua redução unitária de valor objetivo é

$$\bar{z}_i = z_i - c_i = c_B^\intercal B^{-1} a_i - c_i. \tag{98}$$

Lembre-se que $B = [a_1, \ldots, a_i, \ldots, a_m]$ e $B^{-1}B = \mathbf{I} = [e_1, \ldots, e_i, \ldots, e_m]$. Para uma mesma coluna a_i, as linhas serão definidas pelo produto das linhas $1, \ldots, m$ de B^{-1}

com a_i. Ou seja, as demais colunas de B são irrelevantes para a determinação do vetor resultante, e podemos concluir que $B^{-1}a_i = e_i$. Logo,

$$\bar{z_i} = c_B^\intercal e_i - c_i. \tag{99}$$

Mas $c_B^\intercal e_i = c_i$. Portanto, $\bar{z_i} = 0$ para todas as variáveis básicas.

A coluna RHS (*right-hand-side*) apresenta os valores da função objetivo e das variáveis básicas na iteração. Já a coluna z expressa o mesmo que a linha zero, exceto pela ausência de variáveis não-básicas. Com isso, descrevemos toda a estrutura do tableau. Perceba que tudo que envolve as variáveis básicas é obtido trivialmente e que a coluna z é sempre a mesma.

A Tabela 5 mostra outra representação possível para uma construção inicial do simplex tableau, sem considerar as trivialidades expostas pela Tabela 4.

Tabela 5

	z	x_1	$\dots$	x_n	RHS
z	1	$\bar{z}_1$	$\dots$	$\bar{z_n}$	$c^\intercal \bar{b}$
x_1	0				
$\vdots$	$\vdots$		$B^{-1}A$		$\bar{b}$
x_m	0				

A.7.2. Cálculo de nova iteração por pivoteamento

Podemos iterar o simplex tableau por meio de operações matriciais de pivoteamento. Nesta subseção, vamos enunciar e desenvolver todos os passos que justificam o pivoteamento do simplex.

Sejam k o índice da variável de entrada e r o índice da variável de saída. Então o coeficiente y_{rk} é chamado de *pivô* do método simplex. No simplex tableau, y_{rk} pode ser encontrado na linha r, coluna k, como mostra a Tabela 6.

Dito isso, o pivoteamento do simplex é feito através do Algoritmo 7.

Agora vamos justificar cada um dos passos do pivoteamento. A princípio, perceba que a intenção do pivoteamento é transformar x_k em variável básica. Como o Problema (91) organiza todas as variáveis básicas na forma de variáveis de folga, nosso objetivo é fazer com que x_k apareça com coeficiente 1 na linha r do tableau e 0 em todas as demais. Note que estamos incluindo a linha zero, pois, como mostrado no Apêndice A.7.1,

Tabela 6

	z	x_1	$\dots$	x_k	$\dots$	x_n	RHS
z	1	$\bar{z}_1$	$\dots$	$\bar{z}_k$	$\dots$	$\bar{z}_n$	$c^\intercal \bar{b}$
x_1	0	y_{11}	$\dots$	y_{1k}	$\dots$	y_{1n}	$\bar{b_1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_r	0	y_{r1}	$\dots$	$\textcircled{$y_{rk}$}$	$\dots$	y_{rn}	$\bar{b_r}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_m	0	y_{m1}	$\dots$	y_{mk}	$\dots$	y_{mn}	$\bar{b_m}$

Algoritmo 7 Pivoteamento do simplex

1: Dividir a linha r por y_{rk}.
2: Para todo $t \in \{1, \dots, m\} \setminus \{r\}$, atualizar a t-ésima linha subtraindo dela y_{tk} vezes a nova r-ésima linha.
3: Atualizar a linha zero subtraindo dela $\bar{z}_k$ vezes a nova r-ésima linha.

o valor de x_k nesta linha deverá ser zero. Evidentemente, como consequência de estarmos rearranjando o problema, as demais variáveis também terão seus coeficientes alterados de modo a manter as igualdades entre os lados. Esclarecido o objetivo do pivoteamento, sigamos para as justificações.

Passo 1

Queremos que x_k tenha coeficiente 1 na linha r. A razão para tal é simples: x_r é a única variável básica na restrição r. Para que a igualdade continue sendo possível, segue-se que x_k precisa substituir x_r nesta restrição.

Seja $\mathbf{I} = \{1, \dots, m\}$. Então note que a restrição r pode ser equacionada como

$$\sum_{\substack{i \in \mathbf{I} \\ i \neq k}} y_{ri} x_i + y_{rk} x_k = \bar{b}_r. \tag{100}$$

Perceba que basta dividir ambos os lados da equação por y_{rk},

$$\sum_{\substack{i \in \mathbf{I} \\ i \neq k}} \frac{y_{ri} x_i}{y_{rk}} + x_k = \frac{\bar{b}_r}{y_{rk}}, \tag{101}$$

e atingimos nosso objetivo. Note que os coeficientes das demais variáveis também sofrerão alterações neste processo. Por fim, perceba que se $x_r = 0$ nestas condições, x_k será uma variável básica adequada, de acordo com o discutido para se chegar à Equação (95).

Para os passos a seguir, será útil rearranjar a Equação (101) de modo a isolar x_k:

$$x_k = \frac{\bar{b}_r - \sum_{\substack{i \in \mathbf{I} \\ i \neq k}} y_{ri} x_i}{y_{rk}}. \tag{102}$$

Perceba que x_k está representado apenas em termos de variáveis não-básicas, já que os coeficientes das demais variáveis básicas na linha r são todos 0. Logo, podemos afirmar que

$$\sum_{\substack{i \in \mathbf{I} \\ i \neq k}} y_{ri} x_i = 0. \tag{103}$$

Passo 2

Precisamos garantir que x_k tenha coeficiente zero nas demais restrições. Note que podemos escrever estas restrições no formato

$$\sum_{\substack{i \in \mathbf{I} \\ i \neq k}} y_{ti} x_i + y_{tk} x_k = \bar{b}_t, \tag{104}$$

com $t = 1, \ldots, m$ e $t \neq r$.

Pela Equação (102), podemos reescrever esta equação como

$$\sum_{\substack{i \in \mathbf{I} \\ i \neq k}} y_{ti} x_i + y_{tk} \left(\frac{\bar{b}_r - \sum_{\substack{i \in \mathbf{I} \\ i \neq k}} y_{ri} x_i}{y_{rk}} \right) = \bar{b}_t. \tag{105}$$

A Equação (103) nos assegura que $x_k = \bar{b}_r / y_{rk}$. Assim, podemos reescrever a equação atual como

$$\sum_{\substack{i \in \mathbf{I} \\ i \neq k}} \left(y_{ti} - y_{tk} \frac{y_{ri}}{y_{rk}} \right) x_i = \bar{b}_t - y_{tk} x_k. \tag{106}$$

Efetivamente, fizemos exatamente o que queríamos. Retiramos x_k de todas as restrições além da r e, no processo, provamos o passo 2. Para melhor visualização do que acontece na coluna RHS, o lado direito da equação pode ser reescrito como

$$\bar{b}_t - y_{tk}\frac{\bar{b}_r}{y_{rk}}. \tag{107}$$

Passo 3

Resta determinar o cálculo da linha zero. Precisamos partir de uma expressão que relacione os coeficientes de redução unitária do valor objetivo para encontrar seus valores na nova iteração. O Problema (91) supre esta necessidade, dizendo que

$$z = z_0 - \bar{z}_k x_k - \sum_{\substack{i\in\mathbf{I}\\ i\neq k}} \bar{z}_i x_i. \tag{108}$$

Pela Equação (102), podemos reescrever esta equação da seguinte maneira

$$z - z_0 = -\left[\bar{z}_k\left(\frac{\bar{b}_r - \sum_{\substack{i\in\mathbf{I}\\ i\neq k}} y_{ri}x_i}{y_{rk}}\right) + \sum_{\substack{i\in\mathbf{I}\\ i\neq k}} \bar{z}_i x_i\right]. \tag{109}$$

Sabemos que $x_k = \bar{b}_r/y_{rk}$. Rearranjando a equação, temos

$$z - z_0 = -\left[\bar{z}_k x_k - \sum_{\substack{i\in\mathbf{I}\\ i\neq k}} \left(\bar{z}_i - \bar{z}_k\frac{y_{ri}}{y_{rk}}\right)x_i\right]. \tag{110}$$

Esta equação explicita que os coeficientes $\bar{z}_i$ sofrem a alteração especificada pelo passo 3. Note que a próxima iteração começa quando x_k entra na base. Neste momento, $z = z_0$ e, tendo em vista a Equação (110) e o fato de as variáveis que já estavam na base não alterarem o valor objetivo (ver discussão no Apêndice A.7.1), concluímos que $\bar{z}_k = 0$. Na linha r, após o passo 1, o coeficiente de x_k será 1. Desta forma, é possível descrever a alteração de $\bar{z}_k$ como sendo

$$\bar{z}_k - \bar{z}_k\frac{y_{ri}}{y_{rk}}, \tag{111}$$

o que demonstra que o passo 3 também vale para x_k.

Por fim, no caso do valor de RHS, perceba que ele é o próprio z_0, e que z será seu valor na próxima iteração. Isolando z na Equação (110), e tendo em vista que

$$\sum_{\substack{i\in\mathbf{I}\\ i\neq k}} \left(\bar{z}_i + \bar{z}_k\frac{y_{ri}}{y_{rk}}\right)x_i = 0, \tag{112}$$

concluímos que

$$z = z_0 - \bar{z}_k \frac{\bar{b}_r}{y_{rk}}, \tag{113}$$

e, assim, provamos que o passo 3 é adequado para todas as colunas da linha zero. Desta maneira, mostramos que o pivoteamento é capaz de atualizar corretamente todos os dados relevantes para o método simplex a cada iteração.

A Tabela 7 mostra o resultado do pivoteamento aplicado à Tabela 6.

Tabela 7

	z	x_1	$\dots$	x_k	$\dots$	x_n	RHS
z	1	$\bar{z}_1 - \bar{z}_k \frac{y_{r1}}{y_{rk}}$	$\dots$	$\bar{z}_k - \bar{z}_k \frac{y_{rk}}{y_{rk}}$	$\dots$	$\bar{z}_n - \bar{z}_k \frac{y_{rn}}{y_{rk}}$	$c^\intercal \bar{b} - \bar{z}_k \frac{\bar{b}_r}{y_{rk}}$
x_1	0	$y_{11} - y_{1k} \frac{y_{r1}}{y_{rk}}$	$\dots$	$y_{1k} - y_{1k} \frac{y_{rk}}{y_{rk}}$	$\dots$	$y_{1n} - y_{1k} \frac{y_{rn}}{y_{rk}}$	$\bar{b_1} - y_{1k} \frac{\bar{b}_r}{y_{rk}}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_k	0	$\frac{y_{r1}}{y_{rk}}$	$\dots$	$\frac{y_{rk}}{y_{rk}}$	$\dots$	$\frac{y_{rn}}{y_{rk}}$	$\frac{\bar{b_r}}{y_{rk}}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_m	0	$y_{m1} - y_{mk} \frac{y_{r1}}{y_{rk}}$	$\dots$	$y_{mk} - y_{mk} \frac{y_{rk}}{y_{rk}}$	$\dots$	$y_{mn} - y_{mk} \frac{y_{rn}}{y_{rk}}$	$\bar{b_m} - y_{mk} \frac{\bar{b}_r}{y_{rk}}$

Note que a linha que anteriormente representava x_r passa a ser linha de x_k, pois agora é x_k que está na base.

Uma última consideração de particular importância para o Apêndice A.9 diz respeito à obtenção de B^{-1} a partir do tableau. Suponha que um problema de otimização linear do formato do Problema (75) tenha uma submatriz identidade $m \times m$ dentro da matriz de restrições, rearranjando suas colunas. Então, se escolhermos esta submatriz como base inicial para o problema, na primeira iteração teremos $B = B^{-1} = I$. Perceba que isso significa que, quaisquer que sejam os pivoteamentos que realizarmos sobre o tableau para obter outras bases, eles equivalerão ao processo de obter B^{-1} nas colunas da base inicial do problema.

Como veremos no Apêndice A.8.2, sempre podemos configurar nosso problema de modo a partirmos de uma matriz base igual à identidade na primeira iteração.

A.8. Obtenção da solução básica inicial pelo método de duas fases

Existem vários métodos para obter uma solução básica inicial aceitável para um problema de otimização linear. Um método particularmente interessante é o de duas fases, pois este método mistura a adição de variáveis ao problema (similar àquilo que discutimos no Apêndice A.3) com o uso do método simplex já desenvolvido (ver Apêndice A.6).

A.8.1. O que constitui uma solução básica inicial aceitável

A única condição imposta à solução básica inicial é que seja factível. Se conseguirmos garantir isso, teremos um ponto inicial que pertence ao poliedro e é extremo, o que garante que o método simplex funcionará. Do Problema (91), sabemos que $x_B = B^{-1}b$. Logo, para que a solução básica seja factível, é necessário que $B^{-1}b \geq \mathbf{0}$.

Lembre-se do que discutimos sobre o vetor b no Apêndice A.3. Conseguimos garantir, pelos métodos apresentados, que $b \geq \mathbf{0}$ para qualquer que seja o problema inicialmente formulado. Desta maneira, nossa única preocupação é com os valores da matriz B^{-1}. Evidentemente, se todos os elementos de B^{-1} forem não-negativos, então $B^{-1}b \geq \mathbf{0}$. Todavia, se existirem elementos negativos em B^{-1}, é possível que o vetor resultante contenha elementos negativos, destruindo a não-negatividade do problema.

Perceba que esta discussão não é de grande auxílio com as considerações feitas até aqui. Se tentarmos obter uma solução básica factível procurando por colunas não-negativas da matriz A, teremos que fazer, no pior dos casos, n buscas. No entanto, é possível que esta pesquisa seja em vão, se o número de colunas não-negativas de A for menor do que m. Neste caso, se tentarmos recorrer a um teste de todas as permutações possíveis de colunas de A, estaremos retornando ao problema representado pela Equação (30), anulando todo o progresso teórico desenvolvido nesta seção.

Existe uma maneira trivial de resolver este problema. Lembre-se que adicionamos variáveis de folga a um problema para transformá-lo para sua forma canônica. A toda variável de folga que adicionamos está associado um vetor dos coeficientes desta variável em cada restrição. Este vetor é igual a e_i ou $-e_i$, sendo e_i um vetor de m elementos. Assim, se todas as restrições do problema na forma padrão eram de menor igual, com $b \geq \mathbf{0}$, então a submatriz de A constituída das colunas das variáveis de folga em ordem crescente de índice é uma matriz identidade, e, portanto, se escolhermos as variáveis de folga como variáveis básicas iniciais, teremos $B = I$, e segue-se que $B^{-1}b = b$.

Perceba que a matriz identidade é uma excelente escolha de matriz base inicial, pois, além de só ter elementos não-negativos, ela torna triviais todos os cálculos envolvendo a inversa da base. Porém, nem sempre é tão simples obter esta matriz. Considerando

$b \geq \mathbf{0}$, existem duas situações em que as variáveis de folga não produzirão uma submatriz identidade no problema. O primeiro caso é se pelo menos uma das restrições do problema for de igualdade. Neste caso, não é adicionada variável de folga à restrição, e não temos as m variáveis de folga necessárias para gerar a submatriz identidade. O segundo caso é se pelo menos uma das restrições for de maior igual. Neste caso, a variável de folga correspondente deverá ser subtraída da inequação, em vez de ser adicionada, e a submatriz resultante não será identidade.

A.8.2. Obtenção da submatriz identidade por variáveis artificiais

Suponha o caso em que as variáveis de folga introduzidas a um problema na forma padrão não gerem uma submatriz identidade nas restrições do problema. Em tal situação, podemos recorrer ao artifício a seguir.

Definição A.12 (Variável artificial)**.** Uma variável é *artificial* se, quando adicionada a um problema de otimização linear na forma canônica, nenhuma das restrições de igualdade vira restrição de desigualdade.

Ou seja, se adicionarmos um vetor de variáveis de folga x_a ao Problema (75), teremos

$$\min\ c^\intercal x \tag{114}$$
$$\text{s.a}\ Ax + x_a = b, \tag{115}$$
$$x, x_a \geq \mathbf{0}. \tag{116}$$

Evidentemente, como as igualdades se mantêm, as variáveis introduzidas por x_a são redundantes e iguais a zero. Por esta razão, é bom adotar um termo para contrastar variáveis artificiais com as variáveis originais do problema. A partir daqui, vamos nos referir às variáveis originais como *variáveis legítimas* do problema de otimização linear.

Note que as variáveis artificiais, assim como as variáveis de folga, são úteis para a obtenção do vértice inicial por estarem presentes em apenas uma restrição cada uma. No entanto, diferentemente das variáveis de folga, as variáveis artificiais são redundantes e, portanto, podemos sempre escolher adicioná-las às restrições, em vez de subtraí-las. Suponha que sejam $k \leq m$ as restrições do problema às quais precisamos adicionar variáveis de folga. Então só precisamos adicionar variáveis artificiais às $m - k$ restrições restantes para que a matriz de restrições A passe a ter colunas que, se rearranjadas, formam uma submatriz identidade.

O Algoritmo 8 é usado pelo `Caique.jl` para adicionar variáveis artificiais ao problema de otimização linear. Este algoritmo depende da matriz X^s de coeficientes das variáveis de folga, retornada pelo Algoritmo 1.

Algoritmo 8 Adicionar variáveis artificiais ao problema (Implementação)

1: criar uma matriz X^a de dimensões $m \times 0$, que representa os coeficientes das variáveis artificiais nas restrições
2: **se** não tiverem sido adicionadas variáveis de folga ao problema **então**
3: definir $X^a \leftarrow I$ e pôr todas as variáveis artificiais na base, em ordem crescente de índices
4: **retornar** X^a
5: definir $i \leftarrow 1$, representando a linha atual da matriz das variáveis de folga, X^s
6: definir $j \leftarrow 1$, representando a coluna atual da matriz X^s
7: definir m_s como o número de linhas de X^s
8: definir n_s como o número de colunas de X^s
9: **enquanto** $i \leq m_s$ **fazer**
10: **se** $X^s_{ij} = 1$ e $j \neq n_s$ **então**
11: redefinir $j \leftarrow j + 1$
12: adicionar a variável de folga à base
13: pular para o passo 19
14: redefinir $X^a \leftarrow [X^a, e_i]$
15: adicionar a variável artificial à base
16: concatenar um zero ao vetor de custos
17: **se** $X^s = -1$ e $j \neq n_s$ **então**
18: redefinir $j \leftarrow j + 1$
19: redefinir $i \leftarrow i + 1$
20: **retornar** a matriz X^a

O funcionamento do Algoritmo 8 pode ser pensado da seguinte maneira: partindo da primeira linha e primeira coluna de X^s, sempre nos depararemos com um de três casos:

(a) O elemento é 1.

(b) O elemento é 0.

(c) O elemento é -1.

No caso (a), a variável de folga da coluna atual já satisfaz a restrição da maneira necessária. Nos casos (b) e (c), a variável de folga não satisfaz a restrição atual e, portanto, precisamos adicionar uma variável artificial à linha atual.

Ademais, nos casos (a) e (b), se não estivermos na coluna n_s da matriz X^s, precisamos nos deslocar para a coluna $j + 1$, pois devemos verificar a possibilidade de o caso

(a) acontecer em $j+1$. Se a coluna j atual for a última, permanecemos nela, pois então o algoritmo se encarregará de adicionar variáveis artificiais às $m_s - i$ linhas restantes para iterarmos.

Em todos os casos, pulamos para a próxima linha ao fim da iteração. Como existem m_s linhas na matriz X^s, o algoritmo termina após m_s iterações.

Perceba que este algoritmo só funciona de maneira apropriada quando as variáveis de folga estão organizadas de acordo com o Algoritmo 1. Para outras organizações de X^s, são geradas mais variáveis artificiais do que o necessário. Por fim, note que, quando utilizado em conjunto com o Algoritmo 1, o Algoritmo 8 adiciona variáveis artificiais e de folga à base exatamente na ordem necessária para que a matriz base resultante seja a matriz identidade, não sendo necessário reordenar a base após o processo.

A.8.3. O problema de primeira fase

A adição de variáveis artificiais às restrições cria um novo problema a ser resolvido. Por definição, as variáveis artificiais devem ser redundantes; no entanto, como adicionamos as variáveis artificiais à base para obter um vértice inicial, elas passam a ser relevantes. Nosso objetivo agora é reduzir todas as variáveis artificiais a zero e retirá-las da base. Para tal, podemos redefinir o vetor de custos, gerando um novo problema de otimização linear a resolver:

$$\min\ x_0 \coloneqq \mathbf{1}^\intercal x_a \tag{117}$$

$$\text{s.a}\ Ax + x_a = b, \tag{118}$$

$$x, x_a \geq \mathbf{0}, \tag{119}$$

onde $\mathbf{1}$ é um vetor composto apenas de elementos 1. Neste novo problema, que constitui a *primeira fase* do método de duas fases, as variáveis artificiais deverão ficar com valor zero e sair da base, pois são as únicas variáveis que apresentam custo. Para resolvê-lo, basta usar o método simplex desenvolvido no Apêndice A.6.

Em geral, o resultado da primeira fase é uma solução básica factível constituída apenas de variáveis legítimas. Como as variáveis artificiais serão não-básicas, elas efetivamente voltam a ser redundantes ao fim da primeira fase. Nestas condições, podemos retornar ao vetor de custos do problema original e retirar as variáveis artificiais do problema. Feito isso, inicia-se a *segunda fase* do método de duas fases, em que novamente utilizamos o simplex, mas desta vez para resolver o problema original a partir da solução

básica factível obtida na primeira fase.

Caso ocorra de uma variável artificial permanecer na base com valor positivo ao término da primeira fase, isso significa que não existe vértice no poliedro do problema original. Pelo Teorema A.2, isso significa que o poliedro contém reta. Pela Definição A.1, isso implica que existem um ponto $y \in \mathbb{R}^n$ e um vetor não-nulo $d \in \mathbb{R}^n$ tais que $y + \lambda d$ pertence ao poliedro para todo escalar λ. Mas isso significa que podemos selecionar λ grande ou pequeno o suficiente para que $y + \lambda d \ngeq \mathbf{0}$, violando a não-negatividade de pelo menos uma variável do poliedro. Logo, o poliedro é vazio; ou seja, não existe solução para o problema.

Pode também acontecer de uma variável artificial permanecer na base com valor zero. Neste caso, temos uma degeneração com um de dois significados possíveis. Esta situação será discutida no Apêndice A.9.

A.9. Solucionando problemas envolvendo degeneração

Em geral, o conceito de degeneração apresentado ao fim do Apêndice A.4 requer uma abordagem especial quando aparece em uma iteração do simplex. Nesta seção, veremos os principais casos de degeneração e como lidar com eles.

A.9.1. Degeneração ao fim da primeira fase

Um caso especial de degeneração acontece quando uma variável artificial permanece na base com valor zero ao fim da primeira fase. Nesta situação, podemos eliminar a variável artificial da base de duas maneiras, dependendo da situação.

Situação 1: Pivoteamento é possível

Seja x_r uma variável artificial de valor zero ao fim da primeira fase, e suponha que existe variável não-básica legítima x_k com $\bar{z}_k \leq 0$ e $y_{rk} \neq 0$. Neste caso, as variáveis estão relacionadas e podem ser trocadas por pivoteamento, pois, mesmo se $y_{rk} < 0$, o pivoteamento mantém a factibilidade, pois a degeneração de x_r significa que ele sai da base com o mesmo valor, e x_k deve entrar na base com o mesmo valor. Logo, nenhuma variável do problema destrói as restrições de não-negatividade ou aumenta o valor objetivo, e o novo ponto é um vértice.

Situação 2: Pivoteamento é impossível

Se nenhum x_k não-básico legítimo satisfaz a condição $y_{rk} \neq 0$, então nenhuma variável legítima do problema tem relação com a variável artificial. Consequentemente, a restrição do Problema (91) em que a variável artificial tem efeito é redundante para o problema de otimização linear. Assim, podemos eliminar esta restrição do problema.

Perceba que estas situações exigem alterações nos processos de teste de razão mínima (Algoritmo 6) e precificação (Algoritmo 4), respectivamente. Podemos reformular estes processos de acordo com os Algoritmos 9 e 10.

Algoritmo 9 Teste de razão mínima adaptado para a primeira fase (Implementação)

definir um vetor vazio q de razões
definir $i \leftarrow 1$
enquanto $i \leq m$ **fazer**
    se existe variável artificial na base com valor zero **então**
        retornar a variável de saída $r \leftarrow i$
    senão, se $y_{ik} \leq 0$ **então**
        ignorar $\bar{b_i}/y_{ik}$
    senão,
        concatenar $\bar{b_i}/y_{ik}$ ao vetor q
retornar a variável de saída r tal que $\bar{b_r}/y_{rk}$ é o menor valor do vetor q

Algoritmo 10 Precificação adaptada para a primeira fase (Implementação)

definir $w \leftarrow c_B^\intercal B^{-1}$, conhecido como *vetor de multiplicadores simplex*
definir a matriz não-base como $N \leftarrow [a_{j(1)}, \ldots, a_{j(m)}]$, onde $j(1), \ldots, j(m)$ são os índices das variáveis não-básicas
definir o vetor de reduções unitárias de custo $\bar{z} \leftarrow N^\intercal w - c_N$
obter a variável não-básica x_k tal que $\bar{z}_k = \max(\bar{z})$
se $\bar{z}_k \leq 0$ **então**
    se é a primeira fase **então**
        se existem variáveis artificiais positivas na base **então**
            retornar que o problema original não é factível
        senão,
            remover as linhas do problema em que as variáveis artificiais aparecem
            remover as variáveis artificiais do problema
    retornar a solução atual, pois ela é ótima
retornar o índice k da variável de entrada

A.9.2. Caso geral de degeneração

Em geral, a degeneração é uma característica problemática para o simplex, requerendo um tratamento especial. Veremos dois casos ilustrativos a seguir.

Exemplo A.13 (Degeneração tratável pelo teste de razão mínima). Considere o problema de otimização linear

$$\begin{aligned} \min \quad & x_1 + x_2 \\ \text{s.a} \quad & x_1 + x_2 \leq 1, \\ & x_2 \leq 1, \\ & x_1, x_2 \geq 0. \end{aligned} \tag{120}$$

Adicionando as variáveis de folga x_3 e x_4 ao problema, temos

$$\begin{aligned} \min \quad & x_1 + x_2 \\ \text{s.a} \quad & x_1 + x_2 + x_3 = 1, \\ & x_2 + x_4 = 1, \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned} \tag{121}$$

Obviamente, a solução ótima do problema requer que x_1 e x_2 estejam fora da base. Suponhamos uma iteração do simplex que esteja na base $x_B = [x_2, x_4]^\intercal$. Disso, temos:

Tabela 8: Iteração 1

	z	x_1	x_2	x_3	x_4	RHS
z	1	0	0	1	0	1
x_2	0	1	1	①	0	1
x_4	0	-1	0	-1	1	0

Tabela 9: Iteração 2

	z	x_1	x_2	x_3	x_4	RHS
z	1	-1	-1	0	0	0
x_3	0	1	1	1	0	1
x_4	0	0	1	0	1	1

Perceba que x_4 é igual a zero na iteração 1, mesmo estando na base – portanto, esta solução básica factível representa um ponto degenerado. Não obstante, o método simplex

identifica a variável correta para sair da base, x_2. A segunda iteração não apresenta potencial de redução do valor objetivo. Concluímos que o ponto é ótimo.

Este exemplo é trivial e não representa o tipo de problema que se deseja resolver na realidade. Problemas reais são grandes o suficiente para que possam existir pontos degenerados nos quais o teste de razão mínima que desenvolvemos trava o simplex, como veremos em seguida.

Exemplo A.14 (Degeneração intratável pelo teste de razão mínima). Um caso de degeneração intratável pelo teste de razão mínima pode ser encontrado em [9], no exemplo 3.6. O problema começa com a Tabela 10.

Tabela 10: Iteração 1

	z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS
z	1	3/4	−20	1/2	6	0	0	0	3
x_5	0	(1/4)	−8	−1	9	1	0	0	0
x_6	0	1/2	−12	−1/2	3	0	1	0	0
x_7	0	0	0	1	0	0	0	1	1

Ao tentar resolver o problema a partir deste ponto, percebe-se que apenas variáveis com valor zero saem da base, de modo que, eventualmente, a Tabela 10 volta a ocorrer em alguma iteração, gerando um ciclo que mantém o simplex preso no mesmo valor objetivo eternamente. Para solucionar este problema, existem muitos métodos de eficiência variável feitos para substituir o teste de razão mínima. O que utilizamos na implementação do `Caique.jl` foi o *método lexicográfico*.

A.9.3. O método lexicográfico

Este método toma como base o teste da razão mínima, acrescentando iterações que permitem que o simplex se desloque em direção a um ponto que reduza o valor objetivo, eventualmente. Tomando o Algoritmo 9 como partida, o Método 11 implementa a regra de lexicografia.

Para entender como este método garante que a degeneração será superada, primeiro precisamos da Definição A.15.

Método 11 Lexicográfico (Implementação)

definir um vetor vazio q de razões
definir $i \leftarrow 1$
enquanto $i \leq m$ **fazer**
    se existe variável artificial na base com valor zero **então**
        retornar a variável de saída $r \leftarrow i$
    senão, se $y_{ik} \leq 0$ **então**
        ignorar $\bar{b_i}/y_{ik}$
    senão,
        concatenar $\bar{b_i}/y_{ik}$ ao vetor q
definir $j \leftarrow 0$
definir $I_j \leftarrow \{r \mid q_r = \min q\}$
enquanto $|I_j| > 1$ **fazer**
    esvaziar q
    definir $i \leftarrow 1$
    enquanto $i \leq m$ **fazer**
        se $i \in I_j$ **então**
            concatenar y_{ij}/y_{ik} ao vetor q
        senão,
            ignorar y_{ij}/y_{ik}
        redefinir $i \leftarrow i + 1$
    redefinir $j \leftarrow j + 1$
    definir $I_j \leftarrow \min q$
retornar o índice r da variável de saída tal que r é o único elemento do conjunto I_j

Definição A.15 (Lexicografia de um vetor)**.** Um vetor x é dito *lexicograficamente positivo* (denotado por $x \succ \mathbf{0}$) se:

1. $x \neq \mathbf{0}$.
2. O primeiro elemento não-nulo de x é positivo.

Também se diz que um vetor x é *lexicograficamente não-negativo* (denotado por $x \succeq \mathbf{0}$) se ele é nulo ou lexicograficamente positivo.

Agora podemos provar que o método lexicográfico promove a prevenção de formação de ciclos em pontos degenerados no método simplex.

Seja $L_1 = [\bar{b}, B^{-1}]$ uma matriz $m \times (m+1)$ pertencente ao tableau de um problema de otimização linear. Suponha que a base inicial seja I e, portanto, $B^{-1} = I$. Suponha também que garantimos que $b \geq \mathbf{0}$. Segue-se então que $L_1 \succ \mathbf{0}$. Nosso objetivo é mostrar que qualquer pivoteamento simplex aplicado sobre L_1 e sobre as matrizes subsequentes gerará matrizes com apenas linhas lexicograficamente positivas.

Chamemos de $\hat{B}$ a nova base após o pivoteamento e $\hat{b} = \hat{B}^{-1}b$. Assim, $L_2 = [\hat{b}, \hat{B}^{-1}]$. Como discutido no Apêndice A.7.2, sendo L_2 o resultado do pivoteamento de L_1, vale que as linhas de L_2, enumeradas por i, serão

$$\left[\bar{b}_i - y_{ik}\frac{\bar{b}_r}{y_{rk}},\ y_{i1} - y_{ik}\frac{y_{r1}}{y_{rk}}, \ldots,\ y_{im} - y_{ik}\frac{y_{rm}}{y_{rk}}\right],\ \text{para } i \neq r, \tag{122}$$

$$\left[\frac{\bar{b}_r}{y_{rk}},\ \frac{y_{r1}}{y_{rk}}, \ldots,\ \frac{y_{rm}}{y_{rk}}\right],\ \text{para } i = r. \tag{123}$$

Devido à condição de pivoteamento $y_{rk} \geq 0$, e graças ao fato de a r-ésima linha ser lexicograficamente positiva antes do pivoteamento, a Equação (123) garante que a r-ésima linha continuará sendo lexicograficamente positiva.

Quanto ao caso $i \neq r$, devemos considerar que $i \in \mathbf{I_0}$ ou $i \notin \mathbf{I_0}$. Primeiro, suponhamos $i \notin \mathbf{I_0}$. Se $y_{ik} \leq 0$, então perceba que podemos reescrever a Equação (122) como

$$[\bar{b}_i,\ y_{i1}, \ldots,\ y_{im}] - \frac{y_{ik}}{y_{rk}}[\bar{b}_r,\ y_{r1}, \ldots,\ y_{rm}]. \tag{124}$$

Os dois termos só podem ser lexicograficamente positivo e não-negativo, respectivamente, o que assegura que todas as linhas serão lexicograficamente positivas. Já no caso em que $y_{ik} > 0$, pela definição de $\mathbf{I_0}$, e como $i \notin \mathbf{I_0}$, concluímos que $\bar{b}_r/y_{rk} < \bar{b}_i/y_{ik}$

e, portanto, que $\bar{b}_i - (\bar{b}_r/y_{rk})y_{ik} > 0$. Da Equação (122), a i-ésima linha só pode ser lexicograficamente positiva.

Agora consideremos o caso em que $i \in \mathbf{I_0}$. Nesta situação, $y_{ik} > 0$ e $\bar{b}_i - (\bar{b}_r/y_{rk})y_{ik} = 0$. Assim, surgem dois casos mutuamente exaustivos como consequência. No caso de $i \notin \mathbf{I_1}$, pela definição de $\mathbf{I_1}$, $y_{i1} - (y_{r1}/y_{rk})y_{ik} > 0$ e, pela Equação (122), a i-ésima linha é lexicograficamente positiva. Já se $i \in \mathbf{I_1}$, temos que $y_{i1} - (y_{r1}/y_{rk})y_{ik} = 0$ e devemos examinar se $i \in \mathbf{I_2}$. O processo continua, podendo ter no máximo $m + 1$ passos. A conclusão é que cada linha de L_2 é lexicograficamente positiva. Assim, podemos estender este método para quantas iterações for necessário, garantindo que todas serão lexicograficamente positivas.

Vamos pensar na linha zero do tableau antes e depois do pivoteamento. Note que

$$[c_B\bar{b}, c_BB^{-1}] - [c_{\hat{B}}\hat{b}, c_{\hat{B}}\hat{B}^{-1}] = \frac{z_k - c_k}{y_{rk}}[\bar{b}_r,\ y_{r1}, \ldots,\ y_{rm}]. \tag{125}$$

Note que $[\bar{b}_r,\ y_{r1}, \ldots,\ y_{rm}]$ é a r-ésima linha de $[\bar{b}, B^{-1}]$ e é, portanto, lexicograficamente positivo. Como $z_k - c_k > 0$ e $y_{rk} > 0$, é evidente que

$$[c_B\bar{b}, c_BB^{-1}] - [c_{\hat{B}}\hat{b}, c_{\hat{B}}\hat{B}^{-1}] \succ \mathbf{0}. \tag{126}$$

Agora finalmente temos o que é necessário para mostrar que o método lexicográfico impede a formação de ciclos. Para tal, basta mostrarmos que o método garante que nenhuma base se repita. Suponha, por absurdo, que uma sequência de bases $B_1, \ldots, B_t$ é gerada, com $B_1 = B_t$. Pela análise anterior, sabemos que

$$[c_{B_j}\bar{b}_j, c_{B_j}B_j^{-1}] - [c_{B_{j+1}}\bar{b}_{j+1}, c_{B_{j+1}}B_{j+1}^{-1}] \succ \mathbf{0},\ \text{com } j = 1, \ldots, t-1. \tag{127}$$

Ou seja, temos as relações

$$[c_{B_1}\bar{b}_1, c_{B_1}B_1^{-1}] - [c_{B_2}\bar{b}_2, c_{B_2}B_2^{-1}] \succ \mathbf{0}, \tag{128}$$
$$[c_{B_2}\bar{b}_2, c_{B_2}B_2^{-1}] - [c_{B_3}\bar{b}_3, c_{B_3}B_3^{-1}] \succ \mathbf{0}, \tag{129}$$
$$\vdots \tag{130}$$
$$[c_{B_{t-2}}\bar{b}_{t-2}, c_{B_{t-2}}B_{t-2}^{-1}] - [c_{B_{t-1}}\bar{b}_{t-1}, c_{B_{t-1}}B_{t-1}^{-1}] \succ \mathbf{0}, \tag{131}$$
$$[c_{B_{t-1}}\bar{b}_{t-1}, c_{B_{t-1}}B_{t-1}^{-1}] - [c_{B_t}\bar{b}_t, c_{B_t}B_t^{-1}] \succ \mathbf{0}. \tag{132}$$

Assim, somando todos os termos à esquerda, temos

$$[c_{B_1}\bar{b}_1, c_{B_1}B_1^{-1}] - [c_{B_t}\bar{b}_t, c_{B_t}B_t^{-1}] \succ \mathbf{0}. \tag{133}$$

Mas perceba que dissemos que $B_1 = B_t$, o que contradiz o resultado obtido para a sequência, pois $B_1 - B_t = \mathbf{0}$, o que nos faria concluir que $\mathbf{0} \succ \mathbf{0}$, o que é absurdo! Portanto, $B_1 \neq B_t$, e, já que o número de bases possíveis é finito, está provado que a regra lexicográfica não repete bases e, desta maneira, converge em um número finito de passos.

A.10. Resolução de problemas de otimização linear com o Caique.jl

Como explicado anteriormente, implementamos um pacote em Julia [10] para resolução de problemas lineares por meio do método simplex. O `Caique.jl` [20], como é chamado, conta com todas as implementações necessárias para resolver os tipos de problema discutidos nesta seção. Neste breve tutorial de uso, usamos a versão `0.1.0` do pacote.

Considere o Exercício 3-6-2 elaborado por Ferris, Mangasarian e Wright [32]:

$$\begin{aligned} \max \quad & 2x_1 + 3x_2 \\ \text{s.a} \quad & -4x_1 + 3x_2 \leq 12, \\ & 2x_1 + x_2 \leq 6, \\ & x_1 + x_2 \geq 3, \\ & 5x_1 + x_2 \geq 4, \\ & x_1, x_2 \geq 0. \end{aligned} \tag{134}$$

Usando o `Caique.jl`, podemos resolver este tipo de problema com o Código 3.

Código 3 Código em Julia do exemplo 3-6-2 usando o `Caique.jl`.

```julia
using Caique

c = [2, 3] #vetor de custos
A = [-4 3; 2 1; 1 1; 5 1] #matriz de restrições
s = [:less, :less, :greater, :greater] #vetor de sinais
b = [12, 6, 3, 4] #vetor de valores do lado direito
lp = LinearProgram(c, A, s, b) #problema linear
solution = solve(lp, type=:max) #solução do problema
println(solution.iB) #índices das variáveis básicas
println(solution.xB) #valores das variáveis básicas
println(solution.z) #valor ótimo da função objetivo
```

A saída deste programa é mostrada em seguida.

Saída 4 Saída do exemplo 3-6-2 usando o `Caique.jl`.

```
[6, 1, 2, 5]
Real[3.799999999999999, 0.5999999999999999, 4.8, 2.400000000000001]
15.599999999999998
```

Em geral, basta criar a matriz A e os vetores b, c e s e passar estes objetos como argumentos para o construtor `LinearProgram`. Este construtor se encarrega de passar o programa para a forma canônica por meio do método `createSlackSubmatrix`, e consegue também gerar as variáveis artificiais necessárias para a primeira fase com o método `cre⌋ateArtificialSubmatrix`.

Feito isso, basta usar o método `solve` para resolver o problema. Este método conta com alguns parâmetros opcionais. O parâmetro `iB` permite que o usuário utilize uma solução básica factível inicial de sua escolha, fazendo com que a segunda fase do simplex seja executada diretamente. O parâmetro `type` permite que o usuário especifique se o problema é de minimização (`:min`, padrão), de maximização (`:max`), ou se quer apenas a solução de primeira fase (`:firstPhase`). O parâmetro `anticycling` permite que o usuário escolha entre usar o método lexicográfico (`true`) ou o teste de razão mínima (`false`). Por fim, o usuário pode definir o número máximo de iterações que deseja que o simplex realize por meio do parâmetro `maxiter`.

B. Códigos de roteirização

B.1. Implementações do PCV

As formulações de PCV discutidas são apresentadas a seguir.

B.1.1. Formulação DFJ

Código 5 Formulação DFJ do PCV.

```
function createTSPModel(C)
    n = size(C, 1)
    model = Model()
    @variable(model, X[1:n, 1:n], Bin)
    @objective(model, Min, sum(C .* X))
    @constraint(model, [i = 1:n], sum(X[i, 1:n]) == 1)
    @constraint(model, [j = 1:n], sum(X[1:n, j]) == 1)
    for S in combinations(1:n)
        if length(S) == n
            break
        end
        @constraint(model, sum(X[S, S]) <= length(S) - 1)
    end
    return model
end
```

B.1.2. Formulação DFJ com lazy constraints

Como mostra Schermer [66], o modelo DFJ pode ser implementado com *lazy constraints* por meio de funções mais avançadas do JuMP que promovem uma comunicação direta com o *solver* durante o processo de resolução do problema de roteirização. Iniciamos com o Código 6, em que adicionamos uma restrição que impede a formação de laços,

$$x_{ii} = 0, \ \forall i \in \mathbf{P}. \tag{135}$$

Em seguida, é implementada a função de remoção dinâmica de subciclos. Esta função começa checando a condição atual do modelo. A *flag* `CALLBACK_NODE_STATUS_INTEGER` indica que o *solver* encontrou uma solução inteira e, portanto, é hora de procurar por possíveis subciclos e adicionar as *lazy constraints* correspondentes. Para que esta função seja chamada pelo *solver*, é configurado um atributo do modelo que associa ao *solver* a função de remoção de subciclos na forma de *lazy constraints*.

A função que encontra subciclos, por sua vez, recebe a matriz de adjacências da solução atual do problema interpretada como um vetor de arestas, percorrendo as arestas para determinar todos os subciclos do problema. Schermer sugere adicionar apenas a restrição que remova diretamente o menor subciclo do problema.

Código 6 Formulação DFJ do PCV com *lazy constraints*.

```
function createDFJTSP(model, C)
    n = size(C, 1)
    @variable(model, X[1:n, 1:n], Bin)
    @objective(model, Min, sum(C .* X) / 2)
    @constraint(model, [i in 1:n], sum(X[i, 1:n]) == 1)
    @constraint(model, [j in 1:n], sum(X[1:n, j]) == 1)
    @constraint(model, [i in 1:n], X[i, i] == 0)
    function callbackRemoveSubcycles(data)
        status = callback_node_status(data, model)
        if status != MOI.CALLBACK_NODE_STATUS_INTEGER
            return
        end
        S = findSubcyclesTSP(callback_value.(data, model[:X]))
        if 1 < length(S) < n
            constraint = @build_constraint(sum(model[:X][i, j] for i in S, j in
            ↪  S) <= length(S) - 1)
            MOI.submit(model, MOI.LazyConstraint(data), constraint)
        end
    end
    set_attribute(model, MOI.LazyConstraintCallback(), callbackRemoveSubcycles)
end
```

Código 7 Função de determinação de subciclos do PCV.

```
function getEdges(X::Matrix{Float64})
    xij = findall(x -> x > 0.5, X)
    edges::Vector{Tuple{Int, Int}} = []
    for x in xij
        push!(edges, (toInt(x[1]), toInt(x[2])))
    end
    return edges
end

function findSubcyclesTSP(edges::Vector{Tuple{Int, Int}}, n::Int)
    unvisited = Set(collect(1:n))
    shortest = collect(1:n)
    while !isempty(unvisited)
        thisCycle, neighbors = Int[], unvisited
        while !isempty(neighbors)
            current = pop!(neighbors)
            push!(thisCycle, current)
            if length(thisCycle) > 1
                pop!(unvisited, current)
            end
            neighbors =
                [j for (i, j) in edges if i == current && j in unvisited]
        end
        if 1 < length(thisCycle) < length(shortest)
            shortest = thisCycle
        end
    end
    return shortest
end
```

B.1.3. Formulação MTZ

Código 8 Formulação MTZ do PCV.

```
function createMTZTSP(model, C)
    n = size(C, 1)
    @variable(model, X[1:n, 1:n], Bin)
    @variable(model, u[2:n], Int)
    @objective(model, Min, sum(C .* X) / 2)
    @constraint(model, [i in 1:n], sum(X[i, 1:n]) == 1)
    @constraint(model, [j in 1:n], sum(X[1:n, j]) == 1)
    @constraint(model, [j in 2:n, i in 2:n, i != j], u[i] - u[j] + n*X[i, j] <=
    ↪  n - 1)
end
```

B.1.4. Formulação GG

Código 9 Formulação GG do PCV.

```
function createGGTSP(model, C)
    n = size(C, 1)
    @variable(model, X[1:n, 1:n], Bin)
    @variable(model, Z[1:n, 1:n] >= 0)
    @objective(model, Min, sum(C .* X) / 2)
    @constraint(model, [i in 1:n], sum(X[i, 1:n]) == 1)
    @constraint(model, [j in 1:n], sum(X[1:n, j]) == 1)
    @constraint(model, [i in 2:n], sum(Z[i, jz] for jz in 1:n) - sum(Z[jnz, i]
    ↪  for jnz in 2:n) == 1)
    @constraint(model, [i in 2:n, j in 1:n], Z[i, j] <= (n - 1)*X[i, j])
end
```

B.2. Implementação do PRVJT genérico

Uma implementação genérica de PRVJT, baseada no modelo apresentado na Seção 5.1, é mostrada a seguir. Esta versão não considera métodos heurísticos.

Código 10 Formulação genérica do PRVJT

```
using Combinatorics

function createVRPTW(model, C, Q, cardK, dem, serv, entries, leaves)
    n = size(C, 1)
    V = collect(1:n)
    Vp = collect(2:n)
    K = collect(1:cardK)
    v(S) = sum(dem[i]/Q for i in S)
    subsets = filter(x -> length(x) >= 2 && length(x) <= length(V) - 1,
    ↪  collect(combinations(V)))
    @variable(model, X[1:n,1:n,1:cardK], Bin)
    @variable(model, b[1:n] >= 0)
    @objective(model, Min, sum(C[i,j].*X[i,j,k] for k in 1:cardK))
    @constraint(model, sum(X[0,j,k] for j in Vp, k in K) <= cardK)
    @constraint(model, [k in K], sum(X[0,j,k] for j in Vp) - sum(X[j,0,k] for j
    ↪  in Vp) <= 1)
    @constraint(model, [i in Vp], sum(X[i,j,k] for k in K, j in V) == 1)
    @constraint(model, [k in K, i in Vp], sum(X[i,j,k] for j in V) -
    ↪  sum(X[j,i,k] for j in V) == 0)
    @constraint(model, [S in subsets], sum(X[i,j,k] for i in S, j in S, k in K)
    ↪  <= length(S) - v(S))
    @constraint(model, [i in V, j in V, k in K], b[i] + serv[i] + C[i,j] -
    ↪  Inf*(1 - X[i,j,k]) <= b[j])
    @constraint(model, [i in V], b[i] >= entries[i])
    @constraint(model, [i in V], b[i] <= leaves[i])
end
```