UNIVERSIDADE DA INTEGRAÇÃO INTERNACIONAL DA LUSOFONIA
AFRO-BRASILEIRA
INSTITUTO DE ENGENHARIAS
ENGENHARIA DA COMPUTAÇÃO
TRABALHO DE CONCLUSÃO DE CURSO

Álvaro da Silva Farias

A Lógica dos Problemas Computacionais

Redenção
12 de Novembro de 2024

Álvaro da Silva Farias

A Lógica dos Problemas Computacionais

Trabalho de Conclusão de Curso apresentada ao Curso de Bacharelado em Engenharia da Computação do Instituto de Engenharias da Universidade da Integração como parte dos requisitos para a obtenção do título de Bacharel em Engenharia da Computação.

Área de concentração: Complexidade Computacional

Orientadora: Prof.ª Dr.ª Márcia Roberta Falcão

Co-orientador: Prof. Dr. Carlos Brito.

Álvaro da Silva Farias
Discente

Márcia Roberta Falcão de Farias
Professora Orientadora

Nicolas de Almeida Martins
Professor Avaliador

Carlos Eduardo Fisch de Brito
Professor Avaliador

Ana Karolinna Maia de Oliveira
Professora Avaliadora

Francicleber Martins Ferreira
Professor Avaliador

Redenção
12 de novembro de 2024

Universidade da Integração Internacional da Lusofonia Afro-Brasileira
Sistema de Bibliotecas da UNILAB
Catalogação de Publicação na Fonte.

Farias, Alvaro da Silva.

F238l

A lógica dos problemas computacionais / Alvaro da Silva Farias. - Redenção, 2025.
85f: il.

Monografia - Curso de Engenharia de Computação, Instituto de Engenharias e Desenvolvimento Sustentável, Universidade da Integração Internacional da Lusofonia Afro-Brasileira, Redenção, 2025.

Orientador: Prof.ª Dr.ª Márcia Roberta Falcão.
Coorientador: Prof. Dr. Carlos Brito.

1. Complexidade computacional. 2. Gadgets lógicos. 3. Lógica. I. Título

CE/UF/BSP CDD 511.3

Dedico este trabalho aos meus amados pais Adécio e Raimunda por todo o apoio e paciência, ao meu irmão Alcides Saraiva que apesar das diferenças sempre esteve ao meu lado dentro e fora da Universidade, à minha amada companheira Vitória Ingrid com quem posso dividir meus melhores e piores dias, ao meu professor Nicolas Martins por me apresentar a área de Teoria da Computação e despertar o desejo de continuar trabalhando e estudando nesta área, aos meus queridos orientadores Márcia Roberta e Carlos Brito por me proporcionarem um momento ímpar de aprendizado e crescimento pessoal e aos meus colegas de longa data que me acompanham desde o início da graduação Jorge e Jardel.

Agradecimentos

Agradeço à UNILAB por me permitir obter esta graduação e proporcionar as ferramentas necessárias para o meu desenvolvimento profissional e pessoal e ainda por me permitir vivenciar momentos tão significativos na minha vida acadêmica compartilhados com tantos outros colegas e estendo este agradecimento a todos os professores que tiveram participação direta ou indiretamente em minha formação.

Agradeço aos professores avaliadores desse trabalho: Dra. Ana Karolinna Maia de Oliveira, Dr. Francicleber Martins Ferreira, Dr. Nicolas de Almeida Martins, por todas as suas recomendações e contribuições para este trabalho.

Agradeço ao professor Tales Paiva por todos os momentos de conversa e descontração que permitiram dias mais leves em períodos mais caóticos.

Agradeço à minha orientadora Dra. Márcia Roberta por toda a confiança, parceria e apoio durante a execução deste trabalho. Agradeço principalmente pelo suporte fora do âmbito acadêmico, em todas as conversas, conselhos e momentos de descontração e a amizade construída.

Agradeço ao meu coorientador Dr. Carlos Brito pelos conhecimentos transmitidos e apoio durante toda esta caminhada, além da amizade que surgiu no desenvolvimento deste trabalho. Agradeço a todos os momentos de "Por que?"que com toda certeza me proporcionaram um crescimento ainda maior como pessoa e profissional.

Agradeço à minha namorada, Vitória Ingrid, pela paciência, cuidado e compreensão, especialmente nos momentos de ausência necessários para a conclusão deste trabalho, assim como nos momentos de estresse e ansiedade, onde seu apoio foi fundamental para superar.

Agradeço à minha mãe, Raimunda (Regina), pelo apoio emocional e a compreensão de que este processo de crescimento, tanto intelectual quanto pessoal, era essencial para um bem maior. Ao meu pai, Adécio, que, assim como

minha mãe, entendeu a importância desse caminho e sempre me apoiou, apesar das dificuldades. Ao meu irmão, Alcides, que, em meio a tudo isso, sempre esteve ao meu lado, compreendendo minhas ausências e me ajudou no dia a dia.

Agradeço também aos meus colegas que me acompanharam nesta trajetória, alguns desde o início e outros que surgiram ao longo do caminho: Jorge Antônio, Carlos Jardel, Ivina Lorena, Samuel Rodrigues, Ítalo Guilherme, João Matheus, Lucas Pinheiro, Francisco Leonardo, Cleilton Sousa, John Muniz, Mateus Lopes, José Dembo, Manuel Finda, Derick Queiroz, Júlio Mário, Antônio Adilson e muitos outros colegas.

“Não existe um caminho para a felicidade. A felicidade é o caminho”

Thich Nhat Hanh

Resumo

Quando estudamos o assunto da NP-completude nos livros ou na internet, as reduções já aparecem prontas, e não é fácil entender como elas foram feitas. É a partir dessa dificuldade que nos orientamos para realizar este trabalho. De acordo com a teoria, existem alguns problemas que são mais representativos para a classe NP: os chamados problemas NP-completos. Em certo sentido, esses problemas podem ser usados como uma linguagem na qual é possível descrever os outros problemas. Essa é uma das intuições por trás da definição de redução de problemas. E isso motiva uma investigação do assunto a partir do ponto de vista da lógica. A ideia chave é que com uma representação em linguagem lógica nós conseguimos descrever qualquer coisa, o que inclui descrever os problemas NP-completos. Isso nos leva ao primeiro objetivo deste trabalho: tentar entender a lógica dos problemas computacionais. E isso acaba levando ao segundo objetivo: tentar entender por que é mais fácil reduzir um problema para a lógica (SAT) do que no sentido contrário. A ferramenta que nós desenvolvemos para alcançar esses objetivos foram os "gadgets"lógicos, que são as formas de induzir o funcionamento lógico que queremos ter. Utilizando os "gadgets"lógicos, nós conseguimos formular reduções mais intuitivas para diversos problemas NP-completos, e muito semelhantes entre si. E por meio delas, nós demos um passo para alcançar o entendimento que motivou o trabalho.

Palavras-chave: complexidade computacional; Problemas NP-completo; gadgets lógicos; Lógica; SAT;

Abstract

When we study the subject of NP-completeness in books or on the internet, the reductions appear ready-made, and it is not easy to understand how they were made. It is from this difficulty that we are guided to carry out this work. According to the theory, there are some problems that are more representative of the NP class: the so-called NP-complete problems. In a certain sense, these problems can be used as a language in which it is possible to describe other problems. This is one of the intuitions behind the definition of problem reduction. And this motivates an investigation of the subject from the point of view of logic. The key idea is that with logic we can describe anything, which includes describing NP-complete problems. This leads us to the first objective of this work: to try to understand the logic of computational problems. And this ends up leading to the second objective: to try to understand why it is easier to reduce a problem to logic (SAT) than in the opposite direction. The tool that we developed to achieve these objectives were the logical gadgets. Using logical gadgets, we were able to formulate more intuitive reductions for several NP-complete problems, which are very similar to each other. And through them, we took a step towards achieving the understanding that motivated the work.
Keywords: computational complexity; NP-complete problems; logical gadgets; Logic; SAT;

Sumário

Capítulo 1

Introdução

Os estudos que geralmente são feitos na área de computação, relacionados a computabilidade dos problemas algorítmicos, buscam categorizar esses problemas pelo grau de suas dificuldades inerentes. É muito comum deparar-se com problemas fáceis e com problemas bem difíceis, mas, como determinar quão difícil é um problema? Durante bastante tempo, vários estudos são realizados com o objetivo de determinar parâmetros que podem definir o nível de dificuldade de um problema e ainda de alguma forma categorizá-los.

A maior parte dos problemas computacionais classificados são definidos a partir de uma máquina de Turing. De maneira resumida, uma máquina de Turing pode ser definida como uma sextupla $M = (Q, \Sigma, q_0, F^+, F^-, \delta)$, onde há o conjunto de estados, o alfabeto, o estado inicial, estados de aceitação, estados de rejeição e a função de transição, respectivamente [Pap96].

Quando se trata de problemas computacionais há várias classes que são estudadas. Em termos de linguagens é possível definir a classe P como os problemas decidíveis em tempo polinomial em uma máquina de Turing determinística (de fita única). Já os problemas NP são aqueles que podem ser verificados em tempo polinomial e em termos de linguagem estar em NP significa que o problema é decidido por uma máquina de Turing não-determinística de tempo polinomial [Sip12].

Em teoria da complexidade computacional tem-se que os problemas NP-completos são os problemas mais difíceis da classe de problemas NP. Um deter-

minado problema "A" em NP está em NPC se somente se todos os problemas NP podem ser reduzidos à ele. A redução polinomial é tida com o principal mecanismo para demonstrar que um problema é NPC [CLR12].

O conceito de problemas NPC foi inicialmente sugerido por Stephen Cook e Leonid Levin em meados dos anos 70. Os autores indicaram que alguns problemas tidos como NP eram de fato NP-completo, logo, se um determinado algoritmo de tempo polinomial fosse apresentado para um problema NPC todos os outros problemas em NP seriam resolvidos em tempo polinomial [Sip12].

A redução polinomial é a principal técnica para investigar problemas NPC pois a mesma consiste em demonstrar que dois problemas são relacionados [GJ79] e apesar disso, muito do que é documentado na literatura no que se diz respeito a tais reduções acabam sendo apresentados de maneira mais direta, não havendo um detalhamento sobre o procedimento de redução realizado. De certa maneira isso responde ao questionamento que leva ao uso da técnica de maneira mais direta, mas deixa uma lacuna dentro de tal análise sobre os passos executados evitando por exemplo uma nova implementação para problemas distintos.

Uma das propostas deste trabalho é apresentar uma investigação sobre os problemas NPC na ótica de suas reduções, partindo do entendimento de um determinado problema e analisando o mesmo com a finalidade de entender quais passos podem ser dados para se obter a redução que muitas vezes é apresentada de forma direta.

Quer dizer, quando a gente se debruça sobre esse assunto, consultando livros ou a internet, a gente encontra um monte de reduções e argumentos de corretude já prontos.

E daí, o sentimento que nos acomete é expresso por perguntas como

- *Como é que se pensou nisso ?*

ou

- *Será que eu conseguiria pensar em algo assim ?*

ou ainda

- *Como é que se pensa sobre isso ?*

Bom, outra proposta desse trabalho é responder essas perguntas.

Mas, voltando ao começo da história, a nossa perplexidade inicial foi superada quando alguém sugeriu: "Bom, em uma redução, um problema funciona como uma linguagem que usamos para descrever outro problema".

Esse é o ponto de vista lógico.

E esse foi o ponto de partida do nosso trabalho — (*o título apareceu logo após o primeiro encontro*)

Quer dizer, com a lógica a gente consegue descrever qualquer coisa.

E com um problema NP-completo a gente também consegue.

Então, em algum sentido, os problemas NP-completos devem conter uma lógica completa dentro deles — (*e os problemas polinomiais devem conter uma lógica incompleta*)

Nesse ponto, o primeiro objetivo do trabalho ficou claro para nós: entender qual é a lógica dos problemas computacionais — (*ou qual é a lógica de cada um deles*)

Mas, isso ainda não é tudo.

Quer dizer, continuando essa conversa, outra pessoa observou: "Vocês já notaram que é bem mais fácil reduzir um problema para a lógica (e.g., SAT) do que fazer a redução no sentido contrário?"

Nesse momento, os outros disseram: "Sim! porque será que isso é o caso?"

E aqui nós temos o segundo objetivo do nosso trabalho.

Bom, o que vem a seguir não é uma resposta completa para essas duas perguntas.

O que nós vamos apresentar é um relato do progresso que nós conseguimos fazer até agora.

O restante desse capítulo faz uma breve revisão da lógica, e uma breve revisão da teoria da NP-completude — (*apresentando o seu problema mais famoso: o* SAT)

O capítulo 2 introduz a ferramenta técnica que nós temos utilizado para realizar a nossa investigação: os *gadgets* lógicos — (*esse é o principal resultado do nosso trabalho*)

No capítulo 3 nós colocamos essa ferramenta para funcionar, apresentando reduções para 4 problemas NP-completos sobre grafos: o CobV, o CInd, o Cliq e o CDom.

Uma leitura preliminar desse trabalho poderia ser interrompida aqui, pois nesse ponto já se pode ter alguma ideia do que nós fizemos — (*e daí pular para a conclusão, onde nós falamos sobre o que nós ainda pretendemos fazer ...*)

Continuando, após alguma experiência com os *gadgets* lógicos, a gente começa a notar que os problemas computacionais já possuem os seus próprios recursos lógicos, por assim dizer.

E a ideia é que explorar esses recursos lógicos pode facilitar tremendamente a tarefa de redução e a prova de NP-completude.

A maneira como isso é feito na prática consiste em formular variantes do problema SAT baseadas nesses recursos lógicos — (*e provar a sua NP-completude*)

E daí, a tarefa se resume a reduzir uma dessas variantes ao problema.

Para tirar vantagem desse esquema, nós apresentamos no capítulo 4 algumas variantes de SAT, e provamos a sua completude utilizando os *gadgets* lógicos.

Outro aspecto interessante da teoria da NP-completude é que apesar da sua aparência superficial distinta, alguns problemas são quase idênticos do ponto de vista lógico.

Assim, faz sentido apresentar a NP-completude dessas famílias em conjunto.

No capítulo 5, nós apresentamos a família do PtT (Partição em Triângulos), que também contém o 3DM e o X3C.

No capítulo 6, nós apresentamos dois problemas sobre triângulos monocromáticos: o TMon-2v e o TMon-2a.

Finalmente, no capítulo 7 nós apresentamos as nossas conclusões.

1.1 Uma breve revisão da lógica[1]

Mas, o que é a lógica?

Bom, a lógica diz respeito a coisas sobre as quais a gente diz: É ou Não é.

Quer dizer, quando se fala em lógica todo mundo costuma pensar em É Verdade e Não é Verdade

— (*ou é falso*)

Mas na realidade pode ser qualquer coisa.

Por exemplo,

- É Verde
- É Selecionado
- É Bonito
- e por aí vai ...

Então, a lógica é o domínio das dicotomias — (*mas, a gente fala sobre isso outra hora ...*)

[1] Nessa seção, nós apresentamos a lógica a partir de um ponto de vista que a torna semelhante aos problemas computacionais que nós vamos ver nos capítulos seguintes.

Daí, uma vez que nós temos essas coisas, o próximo passo é colocá-las em relação.

Quer dizer, uma vez que a gente sabe de uma coisa que ela É (ou Não é), isso pode nos permitir concluir que uma outra coisa É (ou Não é).

Porque?

Ora, porque as duas coisas estão relacionadas logicamente.

E isso significa que existe uma relação lógica que vincula as duas coisas.

A gente pode pensar que a relação lógica mais básica que existe é a *identidade*[2]

$$x \quad = \quad y$$

Se x É, *então* y *também* É

Se x Não é, *então* y *também* Não é

E a ideia é que, se a gente pode olhar para x, então não é preciso olhar para y para saber o que está acontecendo com ele — (*esse é o princípio da representação*)

A seguir, nós temos a relação de *negação*

$$x \quad \text{NEG} \quad y$$

Se x É, *então* y Não é

Se x Não é, *então* y É

A ideia é que isso é diferente do NÃO — (*que é uma relação unária*)

Mas, uma vez que a gente tem os dois no sistema, a gente ganha a oportunidade de dizer a mesma coisa de duas maneiras diferentes

$$x \quad = \quad \text{NÃO}\, y \qquad \equiv \qquad x \quad \text{NEG} \quad y$$

$$\uparrow$$

é a mesma coisa que

— (*o que pode ser útil na prática*)

[2]Não é claro que o sinal '=' seja a melhor forma de representar a relação. Talvez fosse melhor escrever

$$x \quad \text{Id} \quad y$$

Mas, essas duas relações ainda são muito rígidas — (*elas amarram as duas variáveis de maneira muito firme*)

Então, o próximo passo é introduzir a relação de *implicação*

$$x \quad \rightarrow \quad y$$

Se x É, *então* y *também* É

Se x Não é, *então isso nada diz sobre* y

O que é a mesma coisa que a relação de *disjunção* (OU)

$$x \quad \vee \quad y \qquad \equiv \qquad \text{NÃO } x \quad \rightarrow \quad y$$

Mas que a gente pensa de um jeito diferente

$$x \quad \text{OU} \quad y$$

x É *ou* y É

É só com a implicação que o jogo lógico propriamente dito tem início.

Quer dizer, o jogo lógico é o raciocínio.

E o raciocínio consiste em deduzir uma coisa a partir de outra — (*ou propagar informação de um lado para outro*)

A implicação é a relação lógica paradigmática do raciocínio

$$x \quad \rightarrow \quad y$$

Se x É, *então* y *também* É

Quer dizer, saber alguma coisa sobre x pode nos permitir concluir alguma coisa sobre y.

Mas, apenas se x É.

Nesse sentido, a gente pode pensar que a implicação é meia identidade.

E que juntando as duas metades, a gente consegue a identidade inteira

$$x \quad = \quad y \qquad \equiv \qquad x \quad \rightarrow \quad y, \quad \text{NÃO } x \quad \rightarrow \quad \text{NÃO } y$$

A gente também pode pensar que a implicação é meia negação.

E que juntando as duas metades, a gente consegue a negação inteira

$$x \quad \mathsf{NEG} \quad y \qquad \equiv \qquad x \quad \rightarrow \quad \mathsf{NÃO}\ y, \quad \mathsf{NÃO}\ x \quad \rightarrow \quad y$$

Aqui começa a ficar claro que a implicação e o NÃO já bastam para dizer qualquer coisa — (*o que é uma coisa que a gente já sabia — porque?*)

Mas de fato ainda falta uma coisa.

Quer dizer, com a implicação a gente consegue propagar informação de maneira controlada — (*i.e., dependendo do valor que está sendo propagado*)

Mas, a implicação é uma relação binária entre variáveis — (*ou dicotomias*)

E apenas relacionando dicotomias a gente não consegue formar uma tricotomia.

Na prática, isso significa que a gente não consegue descrever qualquer coisa utilizando apenas as variáveis, o NÃO, o NEG e o OU.

Existem diversas maneiras de escapar dessa limitação.

E a solução adotada pela lógica é o esquema de composicionalidade associado aos parênteses

$$\big((x \ \vee \ y) \ \vee \ z\big) \qquad\qquad \big(x \ \rightarrow \ (y \ \rightarrow \ z)\big)$$

Por meio dos parênteses, a gente pode definir relações ternárias

$$\big((x \ \vee \ y) \ \vee \ z\big) \qquad \equiv \qquad \big(x \ \vee \ y \ \vee \ z\big)$$

— (*o* TROU*)*

E uma vez que a gente tem as relações ternárias, a gente consegue construir as relações quaternárias

$$\big((x \ \vee \ y) \ \vee \ z \ \vee \ w\big) \qquad \equiv \qquad \big(x \ \vee \ y \ \vee \ s\big), \quad \big(\mathsf{NÃO}\ s \ \vee \ z \ \vee \ w\big)$$

— (o QUATROU)

Pronto!

Agora nós já temos os recursos suficientes para descrever qualquer coisa com a lógica.

1.2 Uma breve revisão da NP-completude[3]

Não é preciso muito tempo de trabalho com problemas computacionais para descobrir que existem alguns deles que são bem difíceis.

Daí, com mais algum tempo, a gente descobre que existem muitos problemas bem difíceis.

E daí, com ainda mais tempo, a gente nota uma peculiaridade, quer dizer, existe uma grande quantidade deles que começa assim:

- *Será que existe ... ?*

Uma vez que a gente nota isso, já não é difícil formalizar a coisa assim:

Definição: NP é a classe dos problemas de decisão (i.e., SIM/NÃO) que podem ser resolvidos em tempo polinomial por uma máquina de Turing não-determinística.

Quer dizer, a máquina de Turing não-determinística é a máquina que advinha as coisas.

Daí que, ao receber um problema que tem a forma da pergunta acima, ela advinha a coisa que se está perguntando.

Mas, como a coisa pode existir ou não, a máquina precisa verificar se aquilo

[3]Essa seção foi escrita com a suposição de que essa não é a primeira apresentação da teoria da NP-completude ao leitor — (*ou, pelo menos, de que ela não é a última ...*)

que ela advinhou faz sentido ou não — (*i.e., é uma resposta válida para o problema*), e essa verificação leva tempo polinomial — (*esse é o* P *do nome da classe* NP). Dito isso, uma vez que a classe NP foi definida, a gente consegue obter um problema NP-completo de maneira bem prosaica — (*i.e., um problema que, se for resolvido em tempo polinomial, permite obter a solução de qualquer problema da classe* NP *em tempo polinomial*). Vejamos o seguinte problema:

MT-nD: *Será que a máquina de Turing não-determinística M, que executa em tempo polinomial, responde* SIM *ou* NÃO *para a entrada x ?*

Segue disso que torna-se bem fácil ver que MT-nD está na classe NP, porque uma máquina não-determinística N pode adivinhar as "adivinhações"que a máquina M vai fazer durante a sua computação sobre a entrada x. Logo, N simula a computação de M para verificar se a resposta que ela dá é SIM ou NÃO — (*o que leva tempo polinomial*).

Agora, imagine que A é um problema da classe NP. Além disso, imagine que M_A é uma máquina de Turing não-determinística que resolve o problema A em tempo polinomial. Também, imagine que x é uma instância qualquer do problema A, e que M é uma máquina de Turing determinística que resolve o problema MT-nD em tempo polinomial. Então, nós podemos executar a máquina M sobre as entradas M_A e x para resolver o problema A — (*em tempo polinomial*). Uma vez que nós temos o nosso primeiro problema NP-completo, nós podemos obter o segundo por meio de uma redução. Assim, chegamos ao ponto do problema SAT que é definido assim:

- *Será que existe uma atribuição de valores verdade que satisfaz a fórmula*

CNF Φ *?*

A ideia da redução é bem simples, pois quer dizer, SAT é um problema da lógica, e com a lógica é possível descrever qualquer coisa. Nosso objetivo é o de descrever uma computação de uma máquina não-determinística que termina com a resposta SIM — (*por meio de uma fórmula* Φ).

Vejamos a ideia a seguir:

1. Se a fórmula Φ é satisfatível, então a computação existe, e a máquina não-determinística responde SIM
2. Se a fórmula Φ não é satisfatível, então a computação não existe, e a máquina não-determinística responde NÃO

Logo, se a gente tem uma máquina determinística que resolve SAT, então essa máquina resolve MT-nD — (*e todos os outros problemas da classe* NP). Logo, segue o seguinte teorema que já conhecido da literatura:

Teorema: SAT é NP-completo.

A estratégia da lógica para descrever uma computação consiste em descrever o comportamento de cada bit ao longo da computação, o que faz sentido, pois um bit é uma coisa que É ou Não é, em outras plavras:

$$b \text{ É } 1 \quad ou \quad b \text{ Não é } 1$$

Com isso é fácil imaginar que a computação vai acontecer na máquina de Turing, porque na máquina de Turing no máximo um bit da memória muda a cada passo da computação

— (*e isso é fácil de descrever na lógica*).

A seguir, a gente lembra que a nossa máquina de Turing executa em tempo polinomial: n^k unidades de tempo. Logo, a computação só vai ter tempo de visitar no máximo n^k células da memória — (*porque a cabeça de leitura só pode dar um passo a cada unidade de tempo*), e esses são os bits que a computação utiliza, e, além disso, esses são os bits que a lógica precisa descrever. Daí, basta essa ideia que é bem simples:

→ *a variável x_i indica se o i-ésimo bit é 1 ou 0* — *(*por meio dos seus estados É e Não é*)*

Nesse ponto, nós fazemos uma observação bem importante: a lógica não tem a noção de tempo, e por isso, não faz sentido dizer que uma variável lógica muda de valor. No entanto, sem mudança não há computação. Assim, a ideia da lógica é expressar a noção de mudança por meio de duas variáveis. Faremos assim: a ideia é utilizar as variáveis x_i^t e x_i^{t+1} para indicar os valores do i-ésimo bit nos instantes de tempo t e $t+1$. Com isso, para dizer que não muda, basta escrever assim:

$$x_i^t \quad = \quad x_i^{t+1}$$

E dizer que muda, a escrita disso pode se dar por:

$$x_i^t \quad \mathsf{NEG} \quad x_i^{t+1}$$

Surge agora a necessidade de uma variável diferente para representar o bit a cada instante da computação. Daí, como a computação leva tempo n^k, isso vai

resultar em um total de:

$$n^k \cdot n^k = n^{2k}$$

O que não é um problema — (*porque isso é uma quantidade polinomial*). O próximo passo é descrever como a computação começa, e como ela evolui.

Bom, descrever como a computação começa é fácil: basta dizer qual é o valor de cada bit no instante de tempo 0. O valor desses bits é determinado pela entrada x que a máquina recebe e consultando essa entrada, a gente pode escrever a fórmula assim:

$$x_0^0 = \text{É}, \quad x_1^0 = \text{Não é}, \quad x_2^0 = \text{Não é}, \quad x_3^0 = \text{É}, \quad \ldots$$

Agora, é preciso descrever como a computação evolui. Mas, a evolução da computação em um dado instante depende do estado em que a máquina se encontra naquele instante. Isso é algo que ainda não foi representado.

Pode-se imaginar que os estados da máquina são:

$$q_1, \quad q_2, \quad q_3, \quad \ldots, \quad q_m$$

— (*uma quantidade constante*)

Logo, a ideia é fazer algo semelhante ao que foi feito com os bits da memória. Isto é,

→ *a variável y_j^t vai indicar se a máquina se encontra no instante t da*

computação ou não (por meio dos seus estados É *e* Não é*).*

Assim, será preciso determinar o estado da máquina no início da computação:

$$y_0^0 \;=\; \text{É}, \quad y_1^0 \;=\; \text{Não é}, \quad y_2^0 \;=\; \text{Não é}, \quad y_3^0 \;=\; \text{Não é}, \quad \ldots$$

Só falta dizer uma coisa: na máquina de Turing a mudança sempre acontece no lugar em que a cabeça de leitura está posicionada, só que isso é algo que ainda não foi representado. E neste ponto, pode ser feito novamente, o mesmo que foi feito com os bits da memória e os estados da máquina, ou seja:

$\rightarrow$ *a variável* z_ℓ^t *vai indicar se a cabeça de leitura se encontra na posição* ℓ *da fita no instante* t *ou não — (por meio dos seus estados* É *e* Não é*).*

E precisamos dizer como a computação começa:

$$z_0^0 \;=\; \text{É}, \quad z_1^0 \;=\; \text{Não é}, \quad z_2^0 \;=\; \text{Não é}, \quad z_3^0 \;=\; \text{Não é}, \quad \ldots$$

Pronto, nós já se tem todas as variáveis necessárias. Assim, podemos dizer como a computação evolui. Temos que a computação da máquina de Turing evolui por meio da execução das suas regras de transição. Por exemplo:

$$\left(\, \mathtt{q_j},\ \mathtt{0},\ \mathtt{q_h},\ \mathtt{1},\ \rightarrow \,\right)$$

Se a máquina se encontra no estado q_j lendo o bit 0

então ela passa para o estado q_h, escreve o bit 1

e move a cabeça de leitura para a direita

Dito isso, é quase imediato traduzir isso para uma fórmula da lógica

$$\Big(y_j^t \text{ É } \rightarrow \big(z_\ell^t \text{ É } \rightarrow (x_\ell^t \text{ Não é } \rightarrow y_h^{t+1} \text{ É }) \big) \Big)$$

— (*também vai ser obtido fórmulas análogas dizendo que* y_g^{t+1} Não é, *para* $g \neq h$)

$$\Big(y_j^t \text{ É } \rightarrow \big(z_\ell^t \text{ É } \rightarrow (x_\ell^t \text{ Não é } \rightarrow x_\ell^{t+1} \text{ É }) \big) \Big)$$

e

$$\Big(y_j^t \text{ É } \rightarrow \big(z_\ell^t \text{ É } \rightarrow (x_\ell^t \text{ Não é } \rightarrow z_{\ell+1}^{t+1} \text{ É }) \big) \Big)$$

A ideia é que vai ser possível obter fórmulas assim

- para cada estado q_j
- para cada posição da fita ℓ
- e para cada instante de tempo t

A fórmula ficará grande, mas, ainda assim, com tamanho polinomial.

E, finalmente, ainda é preciso dizer que a máquina responde SIM ao final da computação.

Aqui, a gente assume que q_m é o estado de aceitação da máquina. E,

acrescentamos o seguinte estado à fórmula:

$$q_m \ = \ \text{É}$$

Com isso, concluimos que conseguimos a descrição da computação em uma máquina de Turing.

Capítulo 2

Os gadgets lógicos

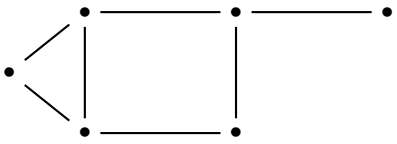
A investigação da lógica dos problemas computacionais é inciada com um problema bastante conhecido: 3-coloração de grafos.

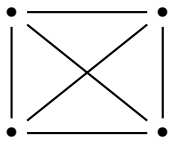
De maneira breve, uma *3-coloração* de um grafo G é uma atribuição das cores Verde, Azul ou Vermelho aos vértices de G, de modo que dois vértices vizinhos não possuem a mesma cor.

Logo, o problema 3Col é definido pela pergunta:

- *Será que o grafo G possui uma 3-coloração ?*

Abaixo está a representação de dois grafos exemplos. Vejamos:





É bem fácil construir uma 3-coloração para o grafo da esquerda.

Mas, é impossível fazer isso para o grafo da direita — (*porque?*)

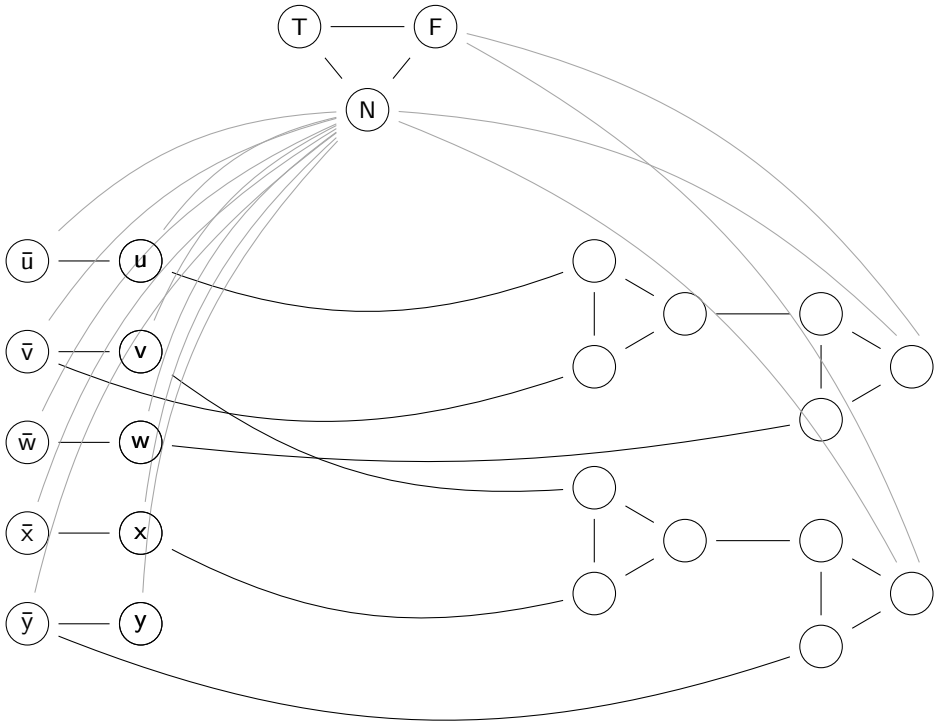


Figura 2.1: 3Col é NPC

Apesar da simplicidade desses exemplos, o problema 3Col é considerado bem difícil na literatura. Veja a seguir o teorema que trata disso:

Teorema: 3Col é NP-completo.

Vemos abaixo o esquema da redução que prova esse fato.

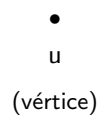
Considere a seguinte fórmula:

$$\Phi \quad = \quad (u \ \vee \ \bar{v} \ \vee \ w) \ \wedge \ (v \ \vee \ x \ \vee \ \bar{y})$$

Nesse ponto, surge a seguinte questão:

- *Qual é a lógica de* 3Col *?*

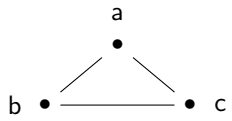
Iniciamos identificando o que É ou Não é. Nesse problema em questão, existem 3 opções:



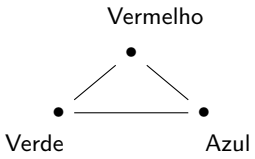
$\Rightarrow$ *u é Verde, u é Azul ou u é Vermelho*

Então, o primeiro passo é restringir a situação para apenas duas opções. Para isso, utiliza-se a seguinte observação:

$\rightarrow$ *Em um triângulo, cada vértice possui uma cor diferente*



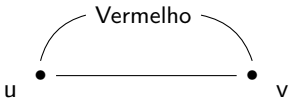
Logo, é possível estipular a cor de cada vértice



— (*i.e., "*Vermelho*" é a cor estipulada para vértice* a). Isso encerra, porque se um vértice u é conectado ao vértice Vermelho, então ele só vai ter duas opções de cores, pela própria definição do problema:

$\Rightarrow$ *u é Verde ou u é Azul*

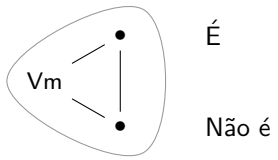
Daí, o próximo passo é relacionar essas coisas. Mas o ponto positivo é que o problema já entrega uma relação de graça, vejamos:



Se u é Verde, v é Azul
Se u é Azul, v é Verde

Quer dizer, essa é a operação de negação (NEG).

E com ela, é possível construir o gadget de uma variável lógica



x

— (*o primeiro gadget lógico obtido*)

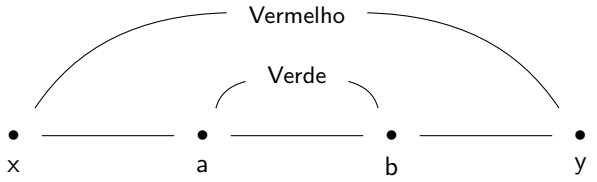
Vamos assumir que a cor Verde determina o estado do gadget, ou seja,

- se o vértice de cima é Verde, então x É
- se o vértice de baixo é Verde, então x Não é

A dificuldade está em que apenas com as variáveis e a operação NEG não conseguimos fazer muita coisa pelo motivo de que as variáveis existem mas não há algo que as relacione entre si. Isso nos leva ao próximo passo que é conseguir o OU ou a implicação

$$(x \vee y) \quad \equiv \quad (\bar{x} \rightarrow y)$$

Quer dizer, a implicação (inferência) é a relação que propaga informação — (*se isso, então aquilo*), e é com ela que o raciocínio lógico começa a funcionar. Essa operação também pode ser vista como a operação OU. Observando o seu funcionamento é possível chegar na seguinte estrutura:

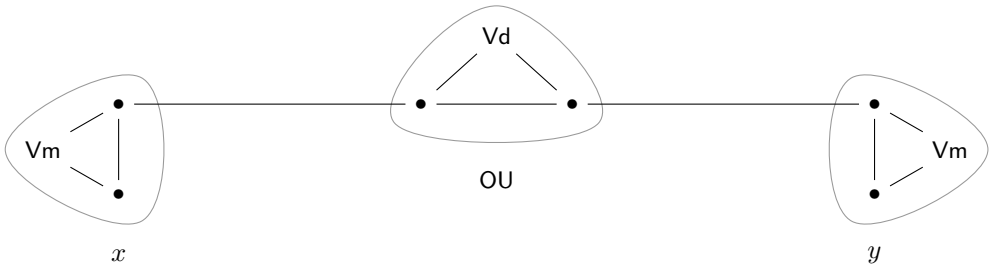


x é Verde ou y é Verde

A ideia é que a e b devem assumir as cores Azul e Vermelho. E o lado do Azul força a variável correspondente a ter a cor Verde. E, no lado do Vermelho, a variável pode ter a cor Verde ou Azul. Seguindo por esse raciocínio vemos três configurações do OU:

x	y
Verde	Azul
Azul	Verde
Verde	Verde

Abaixo nós temos o esquema com as variáveis



— (*esse é o nosso segundo gadget lógico*).

E aqui fica fácil ver que nós temos várias possibilidades de construção. Elas aparecem no esquema a seguir:

$$(x \vee y) \qquad (\bar{x} \vee y) \qquad (x \vee \bar{y}) \qquad (\bar{x} \vee \bar{y})$$

$$(\bar{x} \rightarrow y) \qquad (x \rightarrow y) \qquad (\bar{x} \rightarrow \bar{y}) \qquad (x \rightarrow \bar{y})$$

Outra questão que surge é que apenas com as variáveis e as operações NEG e OU não conseguimos descrever muitas configurações do problema. Dentro da lógica é possível relacionar várias variáveis e não necessariamente apenas duas. Então o próximo passo é conseguir o TROU, que encontra-se esquematizado a seguir:

$$(x \vee y \vee z) \qquad \equiv \qquad (x \vee y) \vee z$$

Quer dizer, o TROU permite obter a *composicionalidade*: É possível montar um OU entre uma variável e uma relação OU do tipo:

$$\underbrace{(x \vee y) \vee z}_{\text{OU}}$$

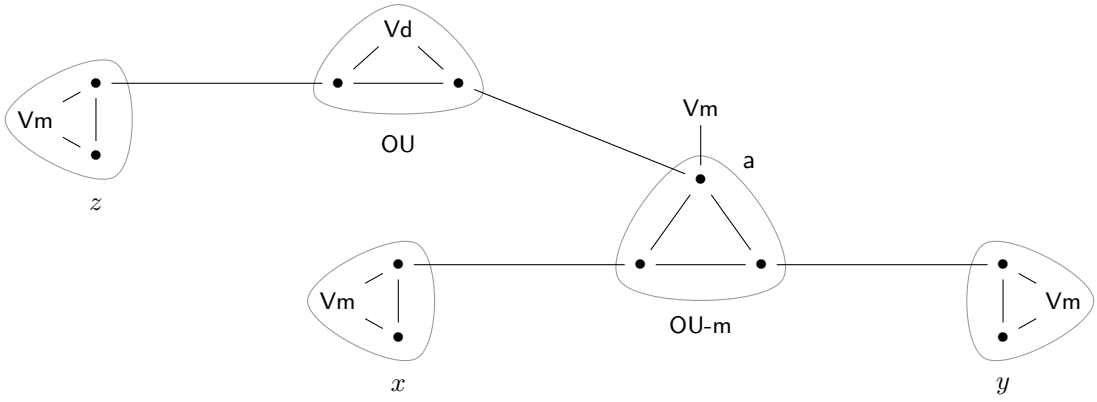
E com essa ideia, pode-se obter o QUATROU (outra ideia que apresentamos no esquema a seguir).

$$\underbrace{\big((x \ \vee \ y) \ \vee \ z\big)}_{\text{OU}} \ \vee \ w$$

Em outras palavras, o TROU introduz no sistema um esquema de montagem, com o qual constroem-se estruturas arbitrariamente complexas, mas ainda é preciso construir o TROU, e com as ideias acima temos algumas intuições de como fazer.

→ *O* TROU *é a relação de* OU *entre uma variável e outra estrutura do* OU

E depois de experimentar um pouquinho, é possível obter a seguinte estrutura



— (*esse é o terceiro gadget lógico*)

A ideia aqui é a seguinte:

1. Na situação em que z = Não é, o vértice a precisa assumir a cor Verde.

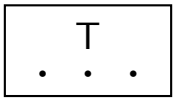
   E isso habilita a lógica do OU entre x e y.

2. Na situação em que z = É, o vértice a fica livre para variar entre Verde e Azul.

   E isso elimina qualquer restrição sobre as cores de x e y.

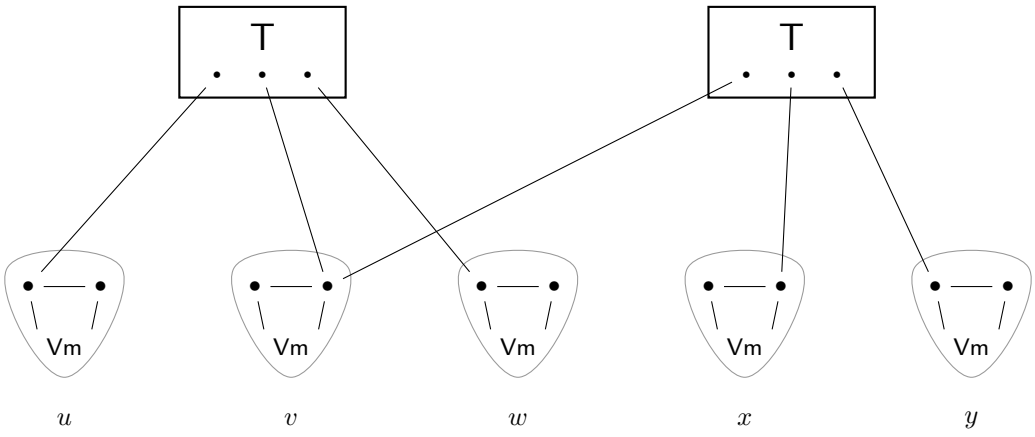
2.1 A NP-completude de 3Col

Agora que os três gadgets lógicos foram obtidos, é bem fácil provar a NP-completude de 3Col. Quer dizer, basta fazer a redução de 3SAT. Para isso, o TROU será representado de maneira esquemática da seguinte forma:

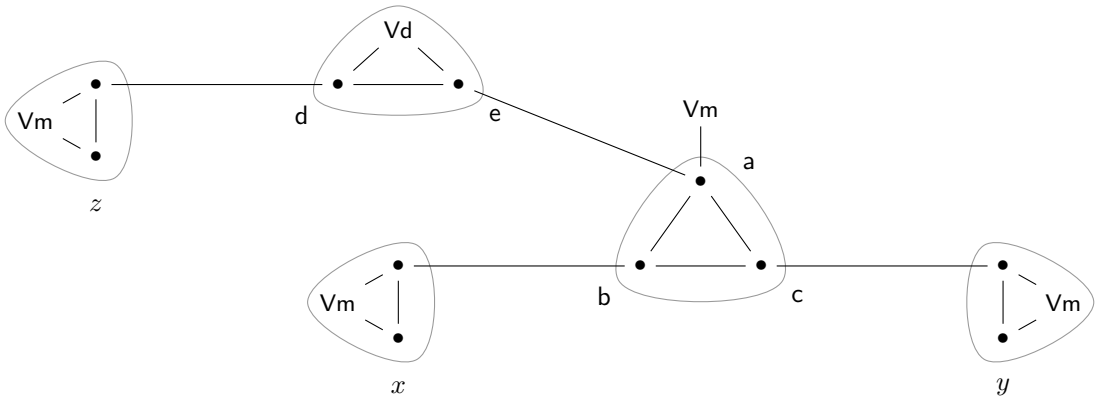


E a redução é representada assim:

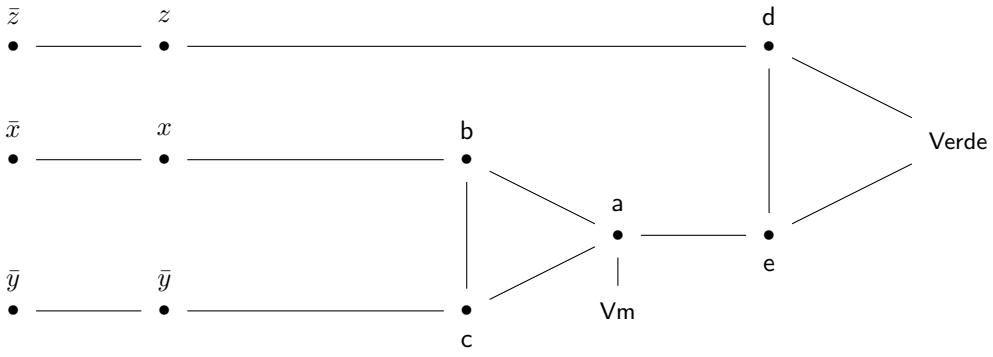
$$\Phi \quad = \quad (u \ \vee \ \bar{v} \ \vee \ w) \ \wedge \ (v \ \vee \ x \ \vee \ \bar{y})$$



Mas, ao apesentar as coisas assim encapsuladas, não é possível visualizar realmente o que está acontecendo. Daí, vamos verificar outra vez o TROU. Note o esquema a seguir:



E agora, é possível reorganizar as coisas assim:



Fazendo isso, o esquema obtido é bem semelhante ao que foi apresentado na figura 2.1 para provar a NP-completude de 3Col.

2.2 O problema da 4-coloração

A seguir, a investigação continua com a variante do problema com 4 cores. Com isso, é possível afirmar o seguinte:

Teorema: 4Col é NP-completo.

Isso é bem fácil de ver, fazendo a redução de 3Col.

Quer dizer, se alguém pergunta

- *Será que o grafo G pode ser 3-colorido ?*

Então, pode-se adicionar um novo vértice a G e conectá-lo a todos os vértices do grafo. E isso leva à segunda pergunta:

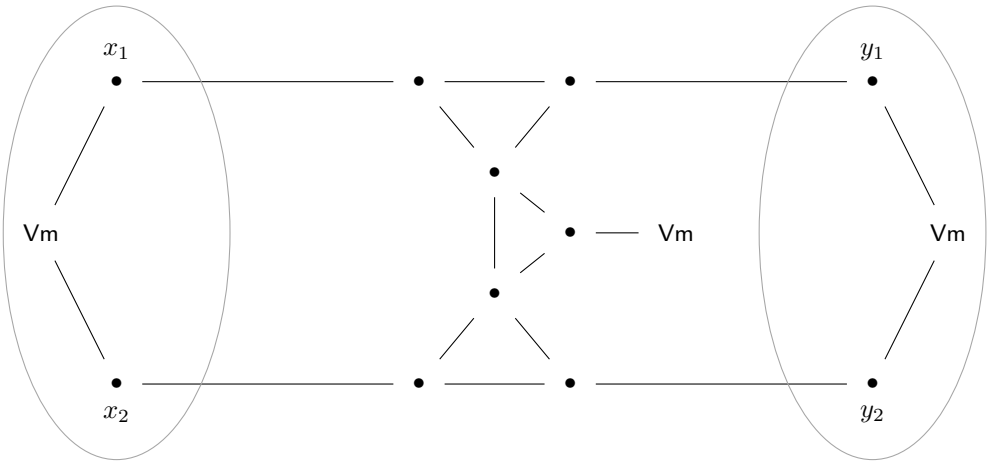
- *Será que esse outro grafo pode ser 4-colorido ?*

A figura abaixo mostra o esquema para a redução:

A ideia é que saber responder a segunda pergunta, então significa saber a resposta para a primeira pergunta.

E isso faz sentido, porque intuitivamente o problema 4Col é mais difícil do que o problema 3Col.

Agora, o mais interessante é que 4Col não é realmente mais difícil do que 3Col — (*módulo* P). Também é possível fazer a redução de 4Col para 3Col. A figura abaixo mostra o esquema da redução:



Quer dizer, nas pontas tem-se os gadgets dos vértices de 4 cores, e no centro o gadget da aresta.

A ideia é que o gadget do vértice é formado por um par, onde cada elemento pode ser Verde ou Azul — (*em um total de 4 possibilidades*).

E o gadget da aresta garante que a coloração de um lado não é igual a coloração do outro lado.

Não é difícil chegar nessa estrutura a partir dos gadgets lógicos[1] — (*se a gente tem em mente o que se quer fazer*).

Mas, essa não é a única maneira de fazer as coisas. Quer dizer, como já foi obtida a lógica de 3Col, e, lembrando que, com a lógica, é possível descrever qualquer coisa.

Para descrever o vértice de 4 cores, temos:

- Utiliza-se 4 variáveis — (*uma para cada cor*);

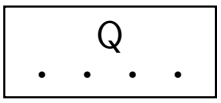
- Implementa-se o OU de todas elas — (*para garantir que uma das cores é*

[1]Note que em cima nós temos um OU, e embaixo nós temos outro OU

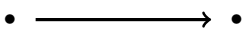
o caso);

- e com isso afirma-se que se uma delas é verdadeira, então as outras são falsas — (*porque o vértice não pode ter mais de uma cor*).

Para fazer essa construção, são utilizados os seguintes componentes:



QUATROU

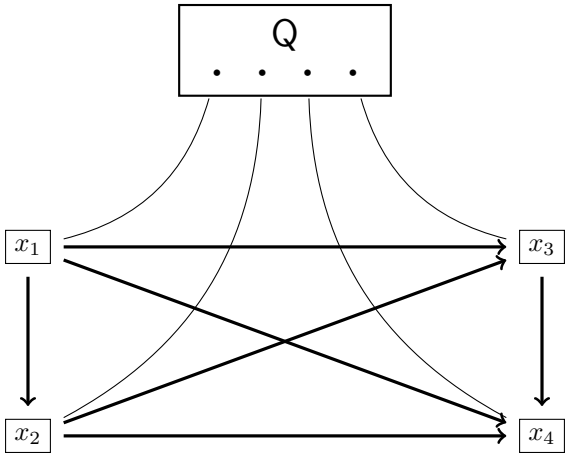


implicação-VF

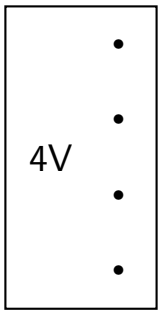
(Se V, então F)

e os gadgets das variáveis.

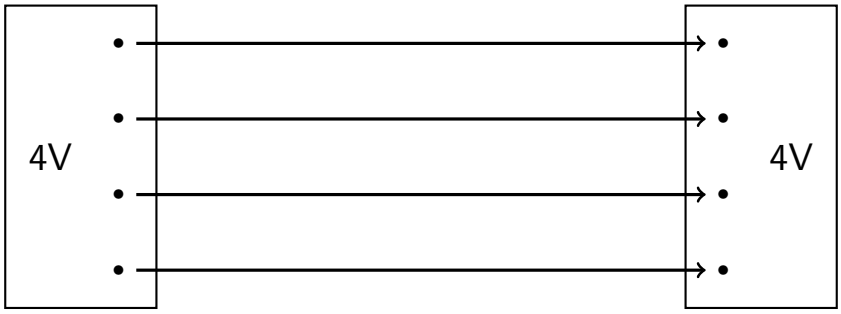
A coisa fica assim



Com isso, é possível representar o vértice de 4 cores dessa maneira:



Logo, o gadget da aresta fica da seguinte forma:



O TROU de 3Col introduz o sistema de montagem que permite obter o QUATROU necessário para o problema 4Col.

Apresentadas as representações que precisávamos, e as ideias de construir essas representações, seguimos para usá-las nos problemas em grafos no próximo capítulo.

Capítulo 3

Alguns problemas em grafos

Neste capítulo, continuamos apresentando a ideia da lógica fazendo uso dos gadgets na representação de alguns problemas de grafos da literatura conhecidos por serem NP-completo. Iniciamos com o problema da Cobertura de Vértices. Em seguida, vemos como essa ideia é aplicada ao problema do Conjunto Independente. Mais a frente, mostramos a mesma maneira de resolução para o problema do Clique. Por fim, apresentaremos o problema do Conjunto Dominante para que fique clara a ideia de como a utilização da estrutura do gadget pode ser aplicada a vários problemas do conjunto NP-completo (NPC).

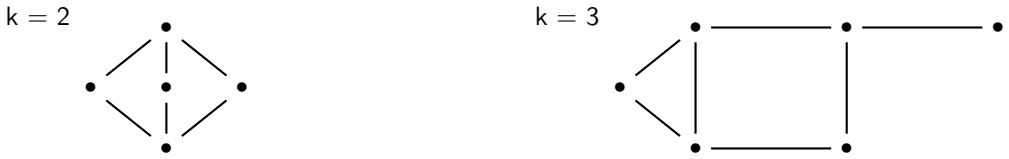
3.1 A lógica da Cobertura de Vértices

A definição para o problema de uma *cobertura de vértices* do grafo G é um subconjunto de vértices C tal que toda aresta do grafo tem ao menos um de seus vértices em C.

Daí, o problema é definido assim

CobV: *Será que o grafo G tem uma cobertura de tamanho k ?*

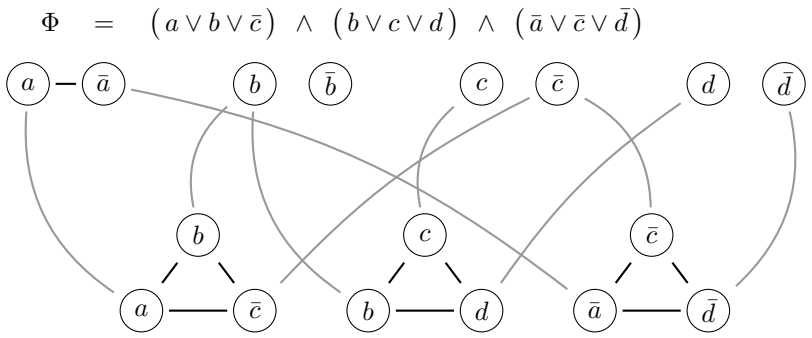
Abaixo tem-se dois exemplos:



Esses exemplos são fáceis, mas em geral o problema é bem difícil

Teorema: CobV é NP-Completo.

Abaixo verifica-se o esquema da redução que prova o teorema



A seguir, é apresentada a abordagem com os *gadgets* lógicos.

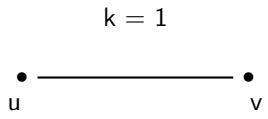
O primeiro passo é definir aquilo que É ou Não é.

E nesse caso, tem-se o seguinte

→ *ou o vértice v é selecionado para a cobertura ou ele não é*

Daí, para obter o *gadget* da variável, é necessário a operação de negação (NEG).

E o mecanismo para essa funcionalidade é a seguinte



Quer dizer,

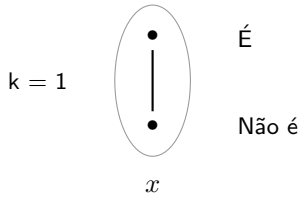
- é preciso selecionar ao menos um vértice para cobrir a aresta
- e o parâmetro k indica que não é possível selecionar mais que um

Logo, exatamente um vértice é selecionado.

E com isso tem-se a relação de negação

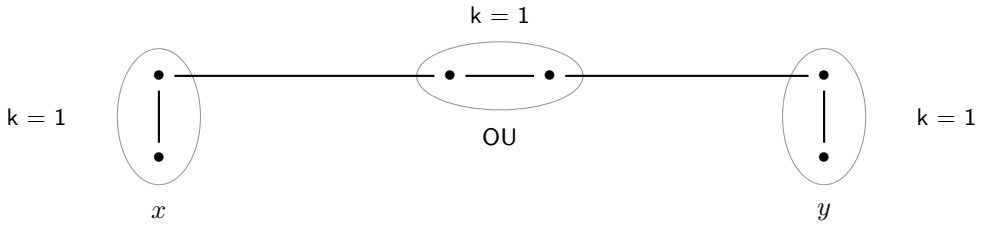
NEG: *Se x é selecionado, então y não é selecionado — (e vice-versa)*

Abaixo é apresentado o esquema para o *gadget* da variável



O próximo passo é construir o *gadget* do OU.

Para isso é utilizada a seguinte esperteza:



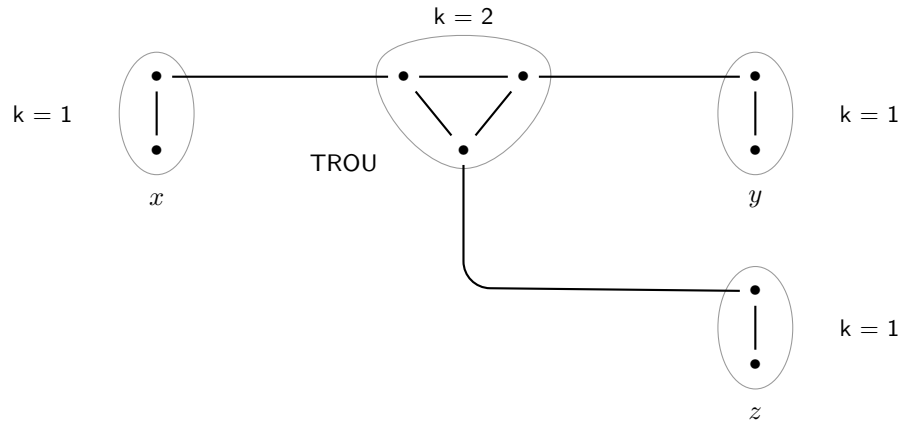
Quer dizer, sabe-se que exatamente um dos vértices da estrutura central vai ser selecionado.

Isso significa que uma das arestas externas vai ficar descoberta.

E isso significa que ao menos uma das variáveis deve assumir o estado É (possivelmente as duas).

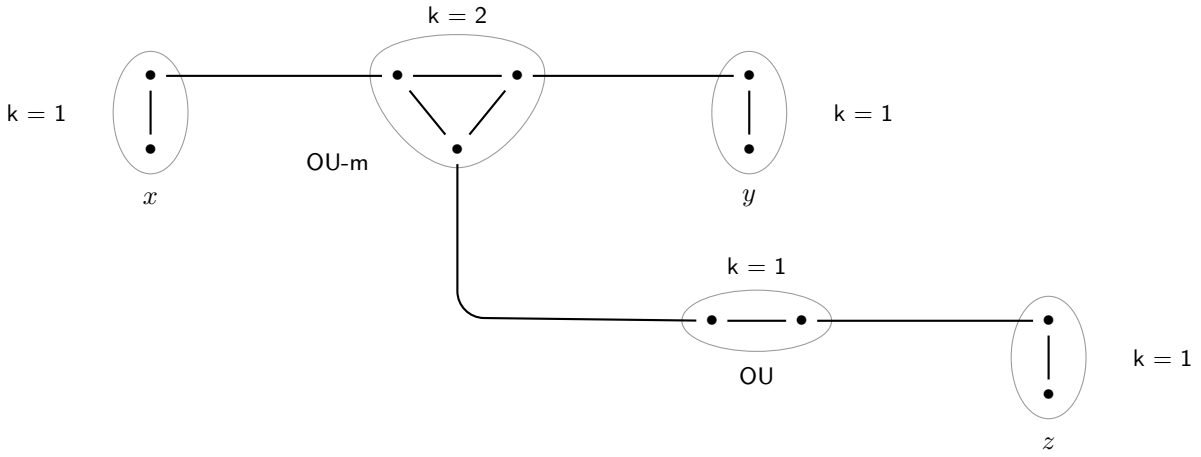
Agora é preciso conseguir o TROU.

E é fácil conseguir isso com uma pequena adaptação do OU



Na prática, esse é o componente que é utilizado na redução do teorema.

Mas, também é possível fazer a construção da seguinte forma



Com isso foi obtido o TROU utilizando um OU e a versão modificada de um segundo OU.

3.2 A lógica do problema do conjunto independente

A definição do problema para um *conjunto independente* do grafo G é um subconjunto de vértices I em que não há nenhum par de vértices conectados por uma aresta. Daí, o problema é definido assim:

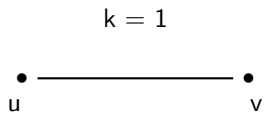
CInd: *Será que existe um subconjunto de vértices I do grafo G de tamanho k tal que nenhum par de vértices em I está conectado ?*

Teorema: CInd é NP-Completo.

A seguir é apresentada a abordagem em termos de *gadgets* lógicos.

O primeiro passo é definir aquilo que É ou Não É.

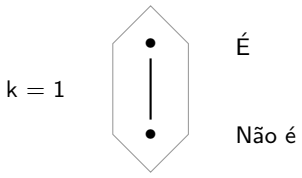
Semelhante ao que foi feito para a *cobertura de vértices*, no problema dos *conjuntos independentes* podemos definir a operação de NEG com a seguinte esperteza:



Quer dizer,

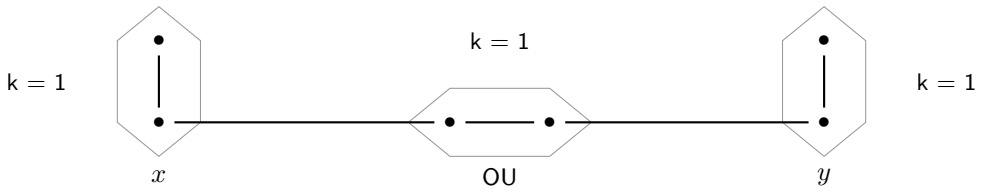
- Em CobV a condição $k = 1$ significa: *No máximo um vértice pode ser selecionado.*
- Em CInd teremos: *Ao menos um vértice deve ser selecionado*

Então o *gadget* da variável (NEG) da lógica de CInd fica assim:



Quer dizer que exatamente um dos vértices do par precisa ser selecionado para ser parte do conjunto independente.

Agora o próximo passo é obtermos *gadget* do OU

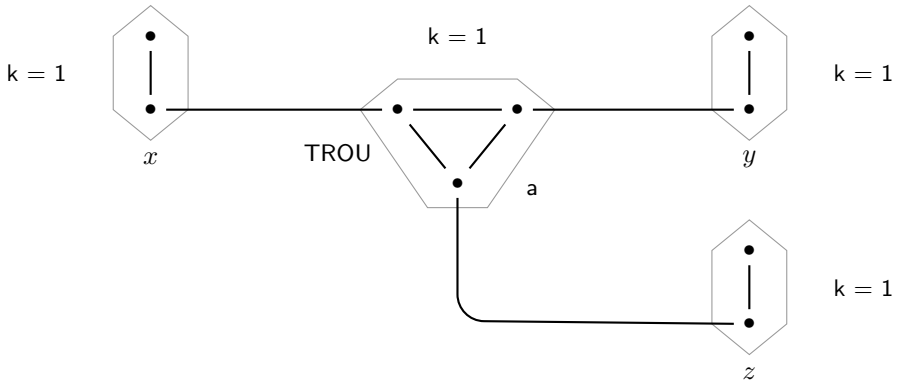


- (Aqui é possível perceber que o problema de Cind é o contrário do problema CobV)

Agora o que falta é apenas a construção do *gadget* do TROU.

A intuição é que o OU entre as variáveis x e y possa ser desligado, o que permite que ambas as variáveis sejam Não É.

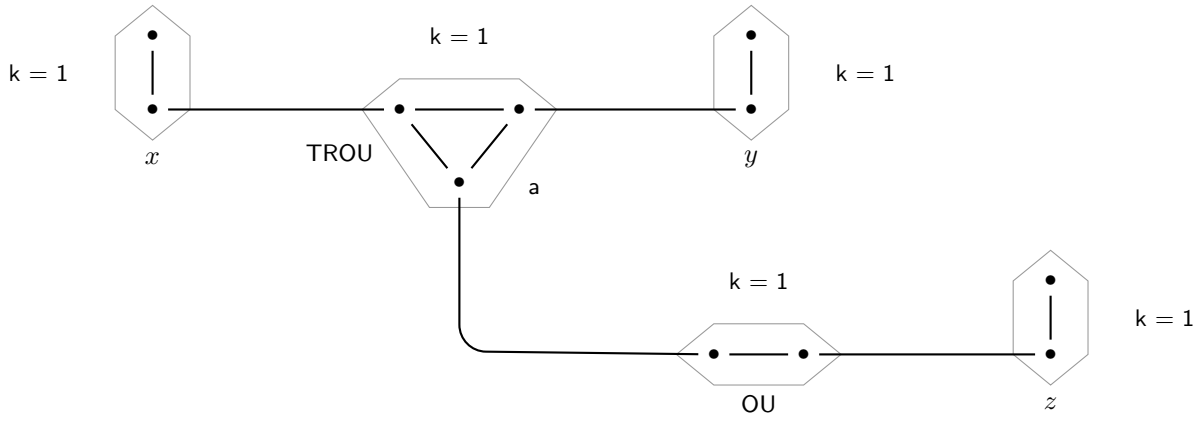
A solução natural é a seguinte:



Análise:

- Quando o vértice a é selecionado, a condição $k = 1$ já é satisfeita, o que impede que os outros dois vértices do *gadget* sejam selecionados e com isso x e y ficam livres para assumirem qualquer valor.
- Quando o vértice a não é selecionado, é preciso escolher um dos outros dois vértices do *gadget* para satisfazer a condição $k = 1$ o que força uma das variáveis (x ou y) assumirem o valor É deixando a outra livre

Ainda é possível reorganizar essa construção para uma forma semelhante ao TROU do CobV



Agora com todos os componentes da lógica de CInd seria bem fácil de realizar a redução de 3SAT caso fossemos apresentá-la aqui, mas vamos continuar apresentando como os outros problemas são representados usando a lógica dos gadgets.

3.3 A lógica do problema do Clique

Um *clique* em um grafo G é um subconjunto de vértices onde cada par de vértices deve estar conectado por uma aresta. Basicamente um *clique* é um subgrafo completo de G.

Daí, o problema é definido assim:

Clique: *Será que existe um subconjunto de vértices C do grafo G de tamanho k tal que todo par de vértices em C está conectado por uma aresta ?*

Teorema: Cliq é NP-Completo.

Bom, esse problema é e não é o contrário do que o problema do *Conjunto Independente*. Quer dizer, ele é o contrário pois um *conjunto independente* no grafo G corresponde a um *clique* no grafo complementar $\bar{G}$ — (e isso já é o suficiente para concluir que Cliq é NP-Completo).

Por outro lado da mesma maneira que CInd, Cliq é um problema de maximização, vejamos:

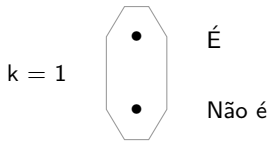
→ *Será que existe um subconjunto (grande) de vértices C com tamanho ao menos k tal que todo par de vértices em C está conectado por uma aresta ?*

Com isso, é esperado que a lógica de Cliq seja semelhante à lógica de CInd.

Na lógica de Cliq é evidente que aquilo que É ou Não É são os vértices do grafo que podem fazer parte da Cliq ou não.

Como apresentado anteriormente em outras seções o *gadget* da variável ou

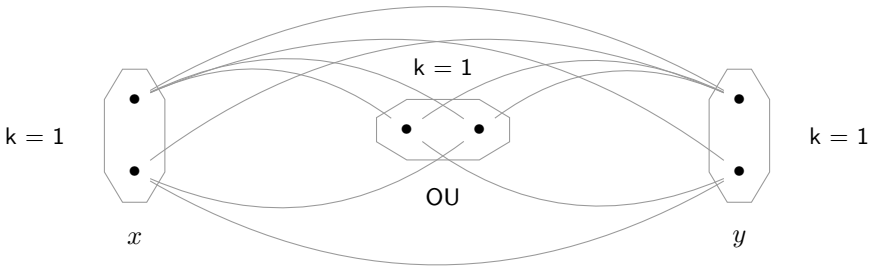
NEG da lógica de Cliq será definida por um par de vértices mas dessa vez, o par de vértice não está conectado por uma aresta.



A intuição é que para satisfazer a condição $k = 1$ obrigatoriamente ao menos um dos vértices deve ser selecionado e como os vértices do *gadget* são desconexos os dois não podem estar juntos dentro do Cliq

Com isso é possível obter a dicotomia da variável

Naturalmente, para a construção do OU podemos partir do complemento da construção apresentada na lógica do CInd



Neste caso, para satisfazer a condição global $k = 3$ é necessário selecionar três vértices e a única maneira de fazer isso é selecionar um vértice de cada componente. A construção da representação segue a seguinte lógica:

- Se x assume o estado É, então y pode estar em Não é e assim escolhemos o vértice da esquerda no componente OU para formar o Cliq satisfazendo a condição global;

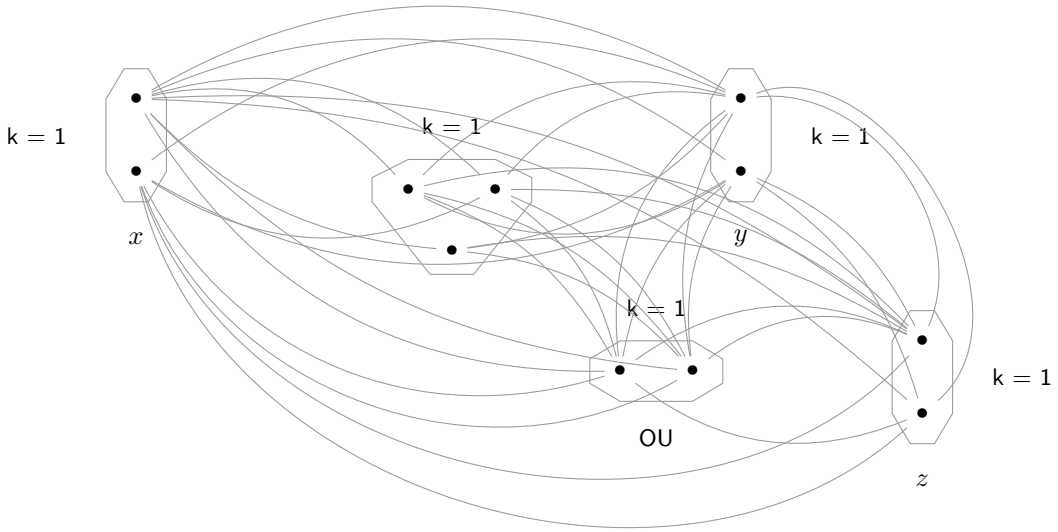
- A situação é análoga para o caso do y É e x Não é;

- Caso as variáveis x e y assumam ambas o estado É isso permite que qualquer vértice do OU seja selecionado;

- E por último, caso as variáveis x e y assumam ambas o estado Não É o *gadget* não funciona, pois os dois vértices Não é estão conectados com vértices diferentes de OU .

Nas observações citadas é possível perceber o papel do comportamento do OU, quer dizer, os vértices de x e y estão todos conectados e a validação de algumas seleções e proibição de outras é feita pelo componente OU.

Com isso, é possível construir o TROU com a mesma intuição da construção do OU - (isto é, contruir o complemento do componente TROU da lógica de CInd).



A intuição dessa vez é semelhante a anterior, onde para satisfazer a condição global de $k = 5$ é necessário selecionar um vértice de cada componente. Podemos descrever o comportamento do mesmo da seguinte forma:

- Se z Não É, então é necessário selecionar o vértice da esquerda do componente OU ao lado de z o que proíbe de selecionar o vértice inferior do OU-modificado forçando a relação de OU entre as variáveis x e y;
- Se z É, abre a possibilidade de selecionar o vértice da direita do componente OU ao lado de z o que permite selecionar o vértice inferior do OU-modificado permitindo as variáveis x e y assumirem qualquer estado.

3.4 A lógica do problema do Conjunto Dominante

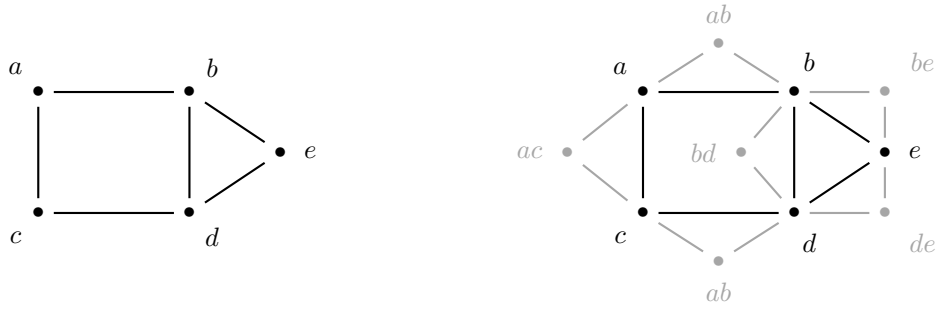
Um *Conjunto Dominante* em um grafo G é um subconjunto de vértices D onde todos os vértices do grafo G ou estão em D ou são vizinhos de algum vértices que está em D.

Daí, o problema é definido assim:

CDom: *Será que existe um subconjunto de vértices D de tamanho k tal que todo vértice do grafo G ou está em D ou tem um vizinho em D ?*

Teorema: CDom é NP-Completo.

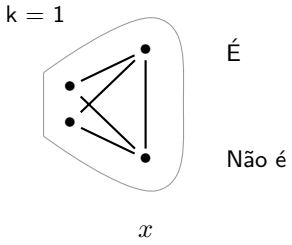
Nos exemplos abaixo é bem fácil identificar os *conjuntos dominantes*



A seguir é apresentada a abordagem em termos de *gadgets* lógicos. Novamente, o primeiro passo é definir aquilo que É ou Não É, e nesse caso tem-se a seguinte ideia:

→ *ou o vértice v é selecionado para o conjunto dominante ou ele não é.*

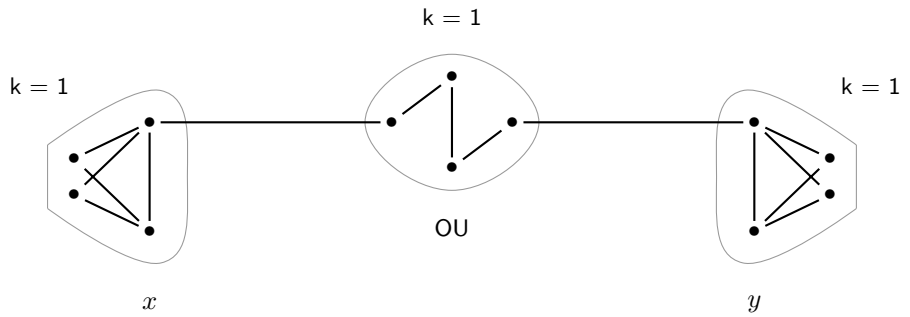
Daí, para obter o *gadget* da variável é necessário definir a operação de negação (NEG), e isso é obtido com a seguinte esperteza:



Quer dizer,

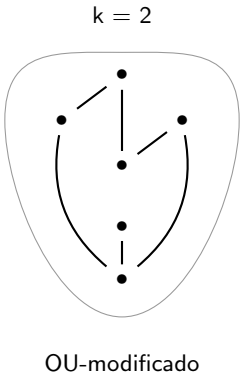
- Em CDom $k = 1$ indica que é necessário pelo menos um vértice para se obter um conjunto dominante no *gadget* da variável;
- Neste caso, os únicos vértices que dominam todos os outros são os vértices mais a direita que indicam os estados É ou Não É.

O próximo passo é obter o *gadget* lógico para o OU. Para isso, tem-se seguinte esperteza:

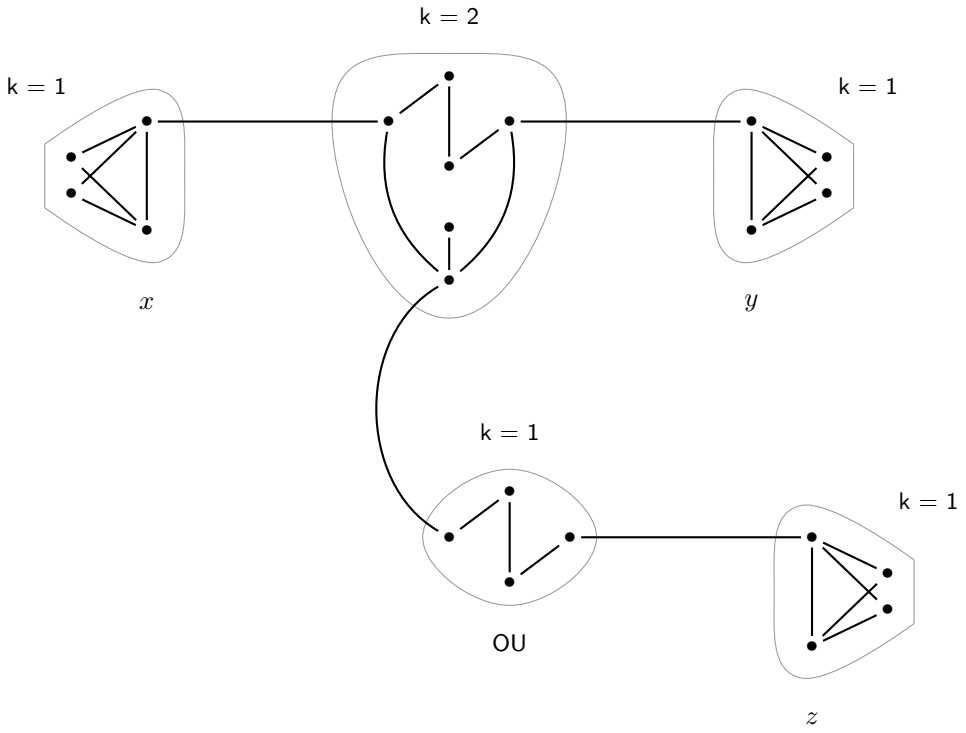


A ideia da construção é que os vértices centrais do OU só podem ser dominados se um deles for selecionado. Isto significa que sempre um dos vértices das extremidades ficará de fora do conjunto dominante. Logo, aquele que não faz parte do *conjunto dominante* deve ser dominado por um dos *gadgets* de variáveis (NEG).

Agora é preciso obter o *gadget* lógico do TROU, e a intuição é a seguinte:



Basicamente, os dois vértices adicionados só podem ser dominados se um deles for selecionado. Isso permite que os *gadgets* de variáveis (NEG) estejam livres para assumir qualquer valor caso o vértice mais a baixo seja selecionado. Com isso é fácil obter o TROU.



Com todos os problemas apresentados nesse capítulo, torna-se clara a ideia que temos como representar os problemas que trabalhamos com a estrutura de gadgets lógicos.

Capítulo 4

Algumas variantes de 3SAT

Neste capítulo, a ideia é verificar como conseguimos representar variantes do problema 3SAT e as semelhanças do que já vimos com a estrutura dos gadegets lógicos. Inicialmente, veremos como fica a lógica para o problema nomeado 1-em-3SAT. Em seguida, apresentaremos a variante nomeada NAE-SAT.

4.1 A lógica do 1-em-3SAT

A variante que nós vamos examinar aqui é a seguinte:

1-em-3SAT: *Será que existe uma atribuição de valores verdade para a fórmula 3CNF* Φ *tal que cada uma de suas cláusulas tenha exatamente um literal verdadeiro ?*

Por exemplo, é fácil ver que isso é possível para a fórmula abaixo

$$\Phi \quad = \quad \big(a \ \vee \ \bar{b} \ \vee \ \bar{d}\big) \ \wedge \ \big(b \ \vee \ c \ \vee \ \bar{d}\big) \ \wedge \ \big(\bar{a} \ \vee \ \bar{c} \ \vee \ \bar{d}\big)$$

$$\texttt{=>} \quad a \ = \ F, \quad b \ = \ F, \quad c \ = \ V, \quad d \ = \ V$$

E também é fácil ver que isso não é possível aqui

$$\Phi \quad = \quad \big(a \ \vee \ b \ \vee \ c\big) \ \wedge \ \big(\bar{a} \ \vee \ b \ \vee \ \bar{c}\big) \ \wedge \ \big(a \ \vee \ \bar{b} \ \vee \ \bar{c}\big) \ \wedge \ \big(\bar{a} \ \vee \ \bar{b} \ \vee \ c\big)$$

Apesar da facilidade desses exemplos, esse problema é bem difícil.

Teorema: 1-em-3SAT é NP-completo.

Para demonstrar esse fato, utiliza-se uma cláusula R 1-em-3 assim

$$R\big(\, x,\, y,\, z\,\big)$$

— (*i.e., a cláusula é satisfeita se somente se exatamente um entre* x, y, z *assume o valor V*).

E daí, a gente nota que é possível construir o TROU utilizando cláusulas R como faremos a seguir:

$$\texttt{TROU}\big(\, x,\, y,\, z\,\big) \qquad \equiv \qquad R\big(\, \bar{x},\, a,\, b\,\big) \ \wedge \ R\big(\, b,\, y,\, c\,\big) \ \wedge \ R\big(\, c,\, d,\, \bar{z}\,\big)$$

Feito isso, surge a pergunta a seguir:

- *Qual é a lógica do* 1-em-3SAT *?*

Bom, isso é um problema da lógica — (*só o que muda é o critério de satisfazibilidade*)

Logo, já tem-se as variáveis booleanas — (*i.e., aquilo que* É *ou* Não é). E também já tem-se o NÃO: $\bar{x}$. Então, ainda falta conseguir o NEG, o OU e o TROU. Mas temos que, o TROU já foi apresentado anteriormente. Queremos mostrar a seguir uma forma obtê-lo por um caminho diferente.

A primeira observação é que é fácil forçar uma variável a assumir o valor F, o que é mostrado em seguida:

$$R\big(x,\,\bar{x},\,y\big) \qquad \Rightarrow \qquad y \;=\; F$$

E depois disso, é fácil forçar uma variável a assumir o valor V

$$R\big(x,\,\bar{x},\,y\big) \;\wedge\; R\big(x,\,\bar{x},\,z\big) \;\wedge\; R\big(y,\,z,\,w\big) \qquad \Rightarrow \qquad w \;=\; V$$

Daí que, a partir de agora, pode-se utilizar as letrinhas V e F sem maiores preocupações.

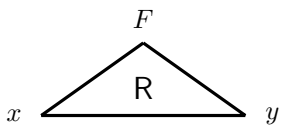
O próximo passo é conseguir o NEG, que é bem fácil

$$R\big(x,\,y,\,F\big) \qquad \Rightarrow \qquad x \;\;\mathsf{NEG}\;\; y$$

Ainda é possível representar o NEG assim

$$x \;\overset{\mathsf{NEG}}{=\!=}\; y$$

ou assim



Daí, é possível utilizar um atalho para construir o OU.

Quer dizer, observa-se que

$$x \ \ \mathsf{OU} \ \ y \qquad \equiv \qquad \mathsf{TROU}\big(x,\, y,\, F\big)$$

E daí, é possível simplificar o TROU da redução acima para chegar na seguinte estrutura:

$$x \ \ \mathsf{OU} \ \ y \qquad \equiv \qquad R\big(\bar{x},\, a,\, b\big) \ \ \wedge \ \ R\big(b,\, y,\, F\big)$$

Mas aqui e percebe-se o NEG obtido anteriormente

$$x \ \ \mathsf{OU} \ \ y \qquad \equiv \qquad R\big(\bar{x},\, a,\, b\big) \ \ \wedge \ \ \underbrace{R\big(b,\, y,\, F\big)}_{\mathsf{NEG}}$$

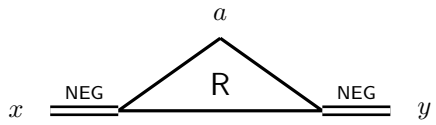
O que permite simplificar a coisa mais ainda

$$x \ \ \mathsf{OU} \ \ y \qquad \equiv \qquad R\big(\bar{x},\, a,\, \bar{y}\big)$$

Análise

- se x, y são ambos F, a cláusula não é satisfeita;
- se um deles é V e o outro F, a assume o valor F;
- se os dois são V, a assume o valor V.

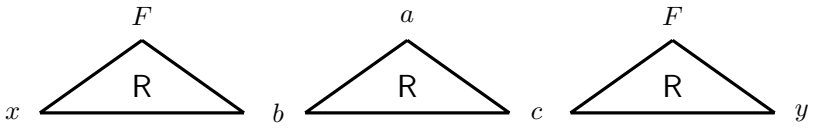
Outra maneira de ver consiste em representar a coisa assim



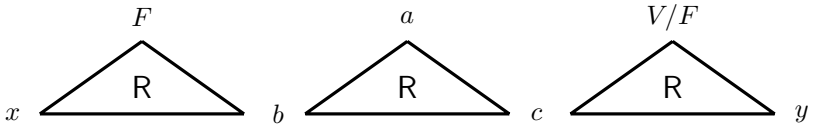
E daí, tem-se o seguinte raciocínio:

- uma das pontas do R é V, e as outras duas são F;
- logo, ao menos uma das pontas de baixo é F;
- logo, ao menos uma das variáveis é V.

Agora é preciso construir o TROU e a estratégia padrão é modificar o OU para adicionar um botão liga-desliga. Mas, o OU atual não pode ser desligado, pois, x, y não podem ser ambos F. Então, a solução é desencapsular os NEG's para ganhar alguma flexibilidade.

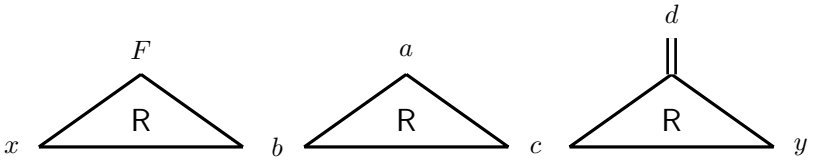


E a flexibilidade é obtida permitindo que um dos F's varie entre V e F.



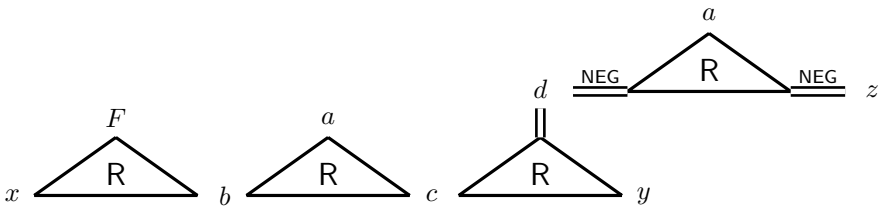
Quer dizer, se o topo do terceiro R for V, então x e y podem ser ambos F, e se o topo do terceiro R é F, x e y podem assumir as outras três configurações de valor. E aí está o botão liga-desliga.

Mas, para tornar a coisa um pouco mais intuitiva, é possível fazer da seguinte forma:



— (*i.e., o* OU *está ligado quando d é V*).

E agora, basta acoplar um segundo OU para obter o TROU.



Vejamos mais outro exemplo na próxima seção.

4.2 A lógica do NAE-SAT

A variante que será analisada nesta seção é NAE-SAT. Vejamos a sua definição.

NAE-SAT: *Será que existe uma atribuição de valores verdade para a fórmula 3CNF* Φ *tal que cada uma de suas cláusulas tenha ao menos um V e ao menos um F?*

Por exemplo, é fácil ver que isso é possível para a fórmula abaixo:

$$\Phi \;=\; \left(a \vee \bar{b} \vee \bar{d}\right) \wedge \left(b \vee c \vee \bar{d}\right) \wedge \left(\bar{a} \vee \bar{c} \vee \bar{d}\right)$$

$$\Rightarrow \quad a = F, \quad b = F, \quad c = V, \quad d = V$$

— (*o mesmo exemplo da seção anterior*).

E também é fácil ver que isso não é possível aqui:

$$\Phi \quad = \quad \big(a \ \vee \ b\big) \ \wedge \ \big(a \ \vee \ \bar{b}\big) \ \wedge \ \big(\bar{a} \ \vee \ b\big) \ \wedge \ \big(\bar{a} \ \vee \ \bar{b}\big)$$

Apesar da simplicidade desses exemplos, esse problema é bem difícil como tentaremos apresentar aqui.

Teorema: NAE-SAT é NP-completo.

Em [KMTB72] a prova desse fato é feita assim:

- Tome a fórmula Φ que foi construída na redução de SAT-Circuito para SAT;
- Nessa fórmula, todas as cláusulas tem tamanho 1, 2 ou 3 literais/ variáveis;
- Adicione cópias da variável z, para fazer com que todas as cláusulas tenha tamanho igual a 3 variáveis;
- E agora argumente que a nova fórmula é NAE-SAT-satisfazível se e somente se o circuito é satisfazível.

Mas, nós vamos perguntar:

- *Qual é a lógica do* NAE-SAT *?*

Bom, isso é um problema da lógica, logo, nós já temos as variáveis booleanas — (*i.e., aquilo que* É *ou* Não é). E também já temos o NÃO: $\bar{x}$. Então agora é preciso conseguir o NEG, o OU e o TROU.

Dessa vez, a gente representa a cláusula NAE-SAT assim:

$$N\big(x, \, y, \, z\big)$$

— (*i.e., essa cláusula é satisfeita se ao menos um de seus literais é V, e ao menos um é F*).

E não é preciso pensar muito para conseguiro NEG, como apresentado abaixo:

$$N(x,\, x,\, y) \qquad \Rightarrow \qquad x \quad \mathsf{NEG} \quad y$$

Como na seção anterior, a gente representa o NEG assim:

$$x \quad \overset{\mathsf{NEG}}{=\!=} \quad y$$

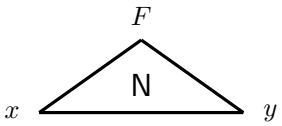
Daí, como x e y vão ter valores opostos, a gente estipula que o valor de x é V, e o valor de y é F.

Agora, a gente pode usar as letrinhas V e F sem maiores preocupações.

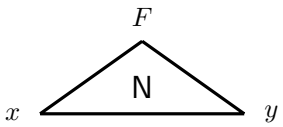
Uma vez que a gente tem o F, é bem fácil conseguir o OU, segue o mesmo:

$$x \quad \mathsf{OU} \quad y \qquad \Rightarrow \qquad N(x,\, y,\, F)$$

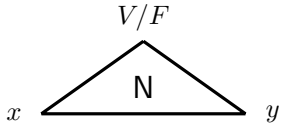
A gente representa o OU assim:



Aqui, já é bem fácil ver o que topo do triângulo é um botão liga-desliga na representação que segue:

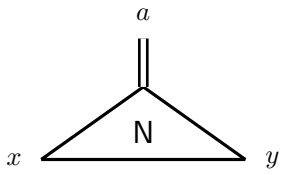


ligado



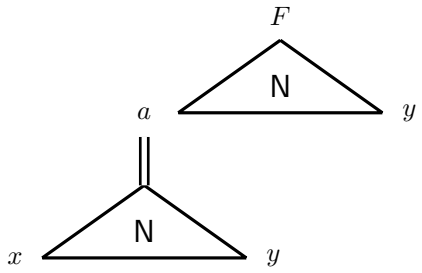
desligado

E para tornar a coisa mais intuitiva, a gente faz assim:



— (*i.e., o* OU *está ligado quando a é V*)

E agora, basta acoplar o segundo OU para obter o TROU



— (*i.e., o* OU *está ligado quando a é V*)

Não é legal?

Capítulo 5

Partição em Triângulos e problemas relacionados

Neste capítulo, vamos representar outro tipo de problema da categoria dos problemas da classe NP-completo. Trata-se dos problemas de Partição em Triângulos, que usamos a sigla PtT. Outro problema que será mostrado é o problema do Emparelhamento-3D (3DM). E, ainda temos o problema do Exact-3-Cover (X3C). A finalidade desse arsenal de problemas em NP-completo é passarmos por problemas com várias características e verificar que nosso estudo de representá-los usando gadgets para tentar reconhecer algum padrão existente na classe dos problemas NPC e ganhar maiores habilidades com a estrutura pensada aqui.

5.1 Lógica do problema da Partição em triângulos

Uma *Partição em Triângulos* é um subconjunto de três vértices C em um grafo G tal que todos os membros do mesmo estão conectados por arestas (formando um clique) e que por sua vez é chamado de triângulo e nenhum triângulo pode compartilhar vértices ou arestas.

Daí, o problema é definido assim:

PtT: *Será que é possível particionar todos os vértices do grafo G em triângulos ?*

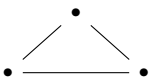
— *(i.e., grupos de 3, onde todos estão conectados por arestas)*

Teorema: PtT é NP-Completo.

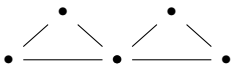
A seguir, nós vamos ver a abordagem com os *gadgets* lógicos para o problema da *partição de triângulos*.

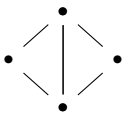
O primeiro passo é definir aquilo que É ou Não É, e no problema PtT nós temos o seguinte:

→ *Ou o triângulo do grafo é selecionado ou não é selecionado*



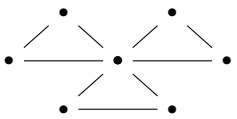
Daí, para obter o *gadget* da variável definimos a operação de negação (NEG)

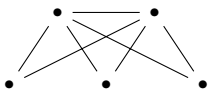




Quer dizer, como os triângulos compartilham vértice ou aresta, exatamente um eles devem ser selecionado para o particionamento.

Essa incompatibilidade (operação de negação) pode ocorrer com mais de dois triângulos, tanto a negação por vértice em comum como a negação por aresta em comum:



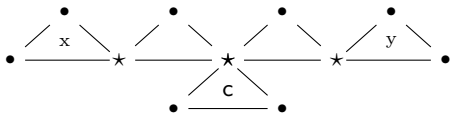


Com isso temos a relação de negação:

NEG: *Se um triângulo é selecionado, então o outro não é selecionado — (e vice-versa)*

O próximo passo para a abordagem dos *gadgets* lógicos é a construção do *gadget* do OU.

E aqui é feita a seguinte esperteza adicionando alguns vértices "estrela"que são obrigados a serem cobertos.

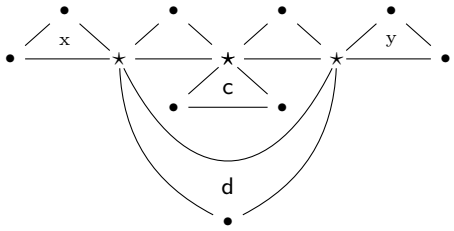


Quer dizer, é fácil perceber que selecionando o triângulo c os triângulos x e y devem ser selecionados, caso contrário, um dos dois ficará de fora do *particionamento*.

Isso significa que ao menos um dos triângulos (x, y) deve assumir o estado É (possivelmente os dois).

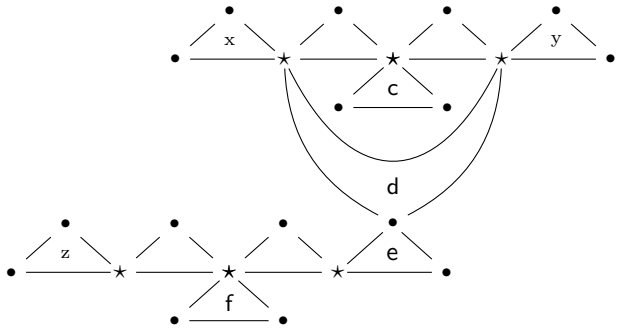
Agora é necessário obtermos o *gadget* do TROU.

E isso é uma tarefa fácil de obter com uma pequena alteração no OU.



Com essa alteração retiramos a restrição de um dos triângulos x, y serem selecionados (ou ambos).

E concluímos a esperteza para o TROU da seguinte forma:



Análise:

1. Se z Não é selecionado, então o triângulo e tem que ser selecionado, o que implica que d Não é selecionado, o que implica que x, y devem satisfazer $(x \vee y)$;

2. Se z É selecionado, então e pode ser ou não ser selecionado, o que implica que d também pode ser ou não ser selecionado, o que elimina qualquer restrição sobre x, y.

Daí, com os *gadgets* lógicos: NEG, OU e TROU é fácil fazer uma redução de 3SAT.

5.2 Lógica do problema do Emparelhamento-3D (3DM)

Para melhor entendimento do problema é possível ilustrar a seguinte situação: Imagine um grupo formado por G *meninas*, B *meninos* e C *cachorros* e além disso imagine a situação em que há um coleção de triplas formadas por *meninas, meninos e cachorros*:

$$(g_1, b_3, c_2),\ (g_2, b_1, c_1),\ (g_1, b_1, c_3),\ ...$$

Essa coleção de tuplas corresponde a um vínculo de família entre os membros.

Daí, o problema pode ser definido da seguinte forma:

> 3DM: *Será que é possível escolher um subconjunto de triplas S de modo que cada menina, menino e cachorro aparece em exatamente uma tripla?*

No exemplo apresentado anteriormente se a primeira tripla é escolhida a terceira não poderá, pois as mesmas compartilham um membro comum (g_1).

Caso a terceira tripla seja selecionada, não será possível selecionar nem a primeira e nem a segunda pois, a terceira tripla compartilha membros com as duas outras.

Com isso nós temos mais um problema computacional a ser analisado.

Teorema: 3DM é NP-Completo.

Para entendermos a lógica do emparelhamento-3D ou 3DM deve-se definir aquilo que É ou Não é para construirmos o *gadget* lógico NEG.

Analisando o problema apresentado, é possível definir aquilo que É ou Não é da seguinte forma:

- *A tripla* $T = (g_i, b_j, c_k)$ *é escolhida ou não é escolhida.*

Com isso é possível verificar também a relação de incompatibilidade que vai descrita a seguir:

- Imagine que b_i faz parte de T_1 e T_2: Se T_1 é escolhido então T_2 não é escolhido.

Mas também há a seguinte situação:

- Imagine que b_i só aparece em T_1 e T_2: Se T_1 não é escolhido então T_2 deve ser escolhido.

Essas relações acabam sendo muito abrangentes, por exemplo: Imagine que $T_1, T_2, T_3, ..., T_n$ são todas as triplas que contém a menina b_i. Então é possível fazer a seguinte afirmação:

- Ao menos uma das triplas é escolhida.

Mas também é possível afirmar o seguinte:

- No máximo uma das triplas é escolhida.

Logo, é possível perceber que a relação entre as triplas é a operação lógica OU-exclusivo:

$$(T_1 \quad \oplus \quad T_2 \quad \oplus \quad ... \quad \oplus \quad T_k)$$

O OU-exclusivo é muito restritivo e para conseguir definir a lógica do problema 3DM são necessárias algumas adaptações para representar as cláusulas do problema e definir a relação entre as variáveis.

Utilizando quatro triplas é possível representar uma variável da seguinte forma:

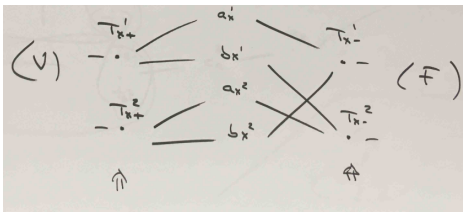


Figura 5.1: variável em 3DM

Ao invés de expressar uma cláusula por meio de cada tripla T_x, T_y, T_z, para cada tripla T que pode ser selecionada, tem-se representantes de uma variável, que assume os valores V ou F e está conectado com as triplas por meio de um cachorro c e conectado com um par (menino, menina) para relacionar os representantes na cláusula.

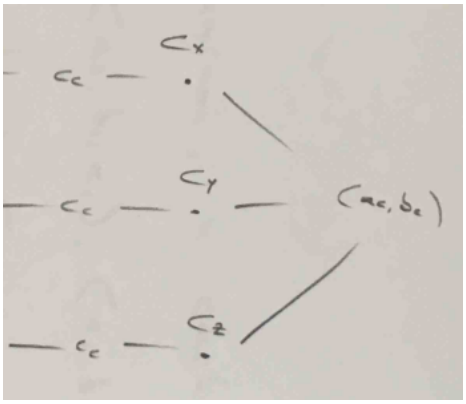


Figura 5.2: cláusula com representantes

E para finalizar a cláusula completa seria a junção entre a construção do *gadget* de representante com o *gadget* da variável:

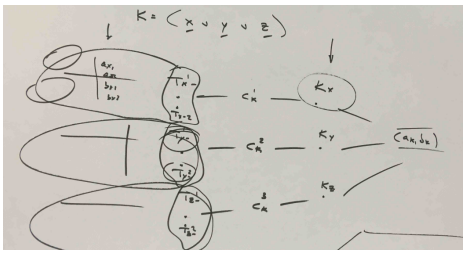


Figura 5.3: cláusula

5.3 Lógica do problema Exact-3-Cover (X3C)

Dado um conjunto X e uma determinada coleção de subconjuntos C, um *exact-cover* é uma subcoleção de conjuntos $C' \subseteq C$ em que cada um dos elementos de X é coberto exatamente uma vez por um subconjunto de C'

Daí, o problema X3C pode ser definido da seguinte forma:

X3C: *Será que é possível cobrir todos os elementos de X utilizando subconjuntos de 3 elementos onde cada elemento só aparece em exatamente um subconjunto ?*

Teorema: X3C é NP-Completo.

Os 3 problemas citados neste capítulo são bem semelhantes e analisando o problema *exact-3-cover*, podemos nos apoiar na redução para o problema das *partições de triângulos*, utilizando elementos semelhantes com base nos *gadgets* lógicos da linguagem dos triângulos.

A intuição é a seguinte, converter o conjunto X com a coleção C em um determinado grafo $G = (V, E)$. Para cada 3-subconjunto podemos implementar um triângulo e que neste caso poderá ter duas soluções dentro do problema PtT que corresponde a selecionar ou não o 3-subconjunto.

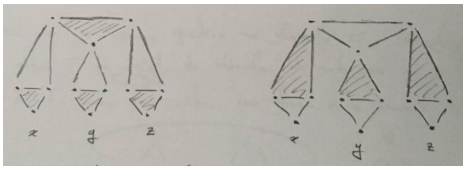


Figura 5.4: 3-subconjunto em PtT

Capítulo 6

Triângulos Monocromáticos

Finalmente, chegamos ao último capítulo de problemas descritos com a estrutura de gadgets e estamos bem satisfeitos em termos manipulado vários problemas da classe NPC. Aqui, vamos ver como ficam os problemas da classe Triângulos Monocromáticos.

6.1 A lógica do problema do Triângulo Monocromático-2V

Um *triângulo monocromático* em um grafo G ocorre quando os 3 vértices conectados por arestas que formam um triângulo estão com a mesma coloração. Daí, o problema pode ser definido da seguinte forma:

> Tmon-2V: *Será que é possível colorir os vértices do grafo G com 2 cores de modo que não exista nenhum triângulo monocromático?*

Teorema: Tmon-2V é NP-Completo.

A intuição que prova a NP-completude desse problema tem como base a redução para uma variante do problema SAT, no caso, o NAE-SAT.

Na variante em questão cada cláusula deve ter ao menos um literal verdadeiro e ao menos um literal falso. A redução desse problema utiliza como artificio de três *gadgets* lógicos:

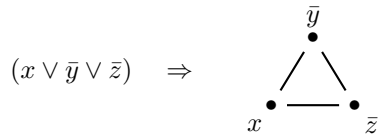
1. O *gadget* das variáveis;
2. O *gadget* das cláusulas;
3. O *gadget* da super-aresta.

O *gadget* da super-aresta é um artifício que garante que um determinado par de vértices devem ter cores opostas e que será representado da seguinte maneira:

Com o *gadget* da super-aresta fica bem fácil construir o *gadget* da variável (NEG):

- (O vértice com a coloração vermelha determina o valor da variável)

O *gadgets* das cláusulas podem ser construídas apenas utilizando um triângulo:



E a redução finaliza quando cada literal das cláusulas são conectados ao literal oposto no *gadget* da variável por uma super-aresta:

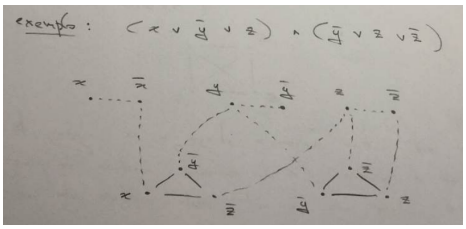
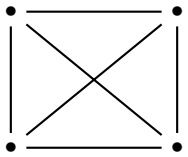


Figura 6.1: Tmon-2V - NAE-SAT

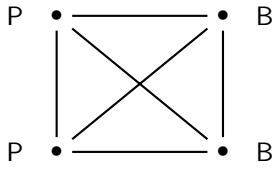
Mas ainda falta apresentar o *gadget* da super-aresta. A propriedade dessa aresta é que para um determinado par de vértices (x, y) no *gadget* tem-se para qualquer coloração válida:

$$C(x) \neq C(y)$$

A ideia é utilizar uma cópia do grafo K_4:



Percebe-se que dois vértices do K_4 devem ter uma cor e os outros dois a coloração oposta - (A única coloração válida em K_4):



Para continuar a construção do *gadget* é necessário encaixar mais um grafo K_4 no par de vértices (B), onde o novo par adicionado deve ser de coloração (P). A ideia é formar uma fileira de cinco estruturas K_4 ligando o último par no primeiro, ou seja, formando um ciclo.

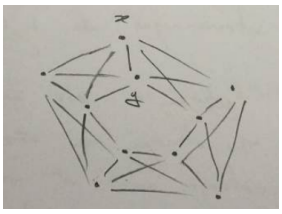


Figura 6.2: super-aresta

Daí que se os primeiros vértices (x, y) tiverem a mesma coloração, não será possível colorir o grafo do *gadget* e para resolver isso, os vértices (x, y) devem ter colorações opostas.

6.2 A lógica do problema do Triângulo Monocromático-2A

O *triângulo monocromático* citado no problema anterior tinha como foco os vértices que formam o triângulo. Nessa variação do problema, a coloração é feita nas arestas. Daí, o problema pode ser definido da seguinte forma:

Tmon-2A: *Será que é possível colorir as arestas do grafo G com 2 cores de modo que não exista nenhum triângulo monocromático?*

Teorema: Tmon-2A é NP-Completo.

Os problemas Tmon-2V e Tmon-2A tem alguma semelhanças e por isso a redução segue basicamente as mesmas etapas:

1. O *gadget* das variáveis;
2. O *gadget* das cláusulas;
3. O *gadget* da super-aresta.

A diferença desta redução é que o *gadget* da super aresta é construído levando em conta a coloração das arestas, ou seja, garantir que um determinado par de arestas devem ter cores opostas:

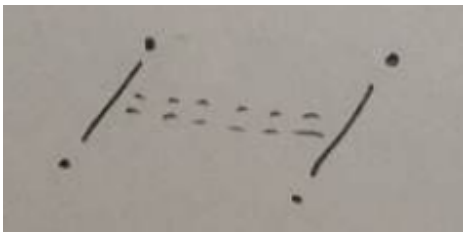


Figura 6.3: super-aresta inicial

Com essa representação fica fácil construir o *gadget* da variável (NEG)

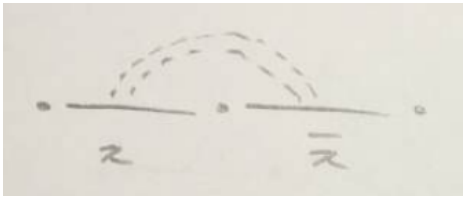
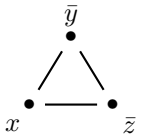


Figura 6.4: NEG - Tmon-2A

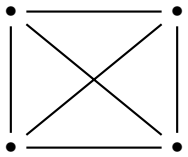
- (A aresta de coloração vermelha determinar o valor da variável)

O *gadget* da variável neste caso fica igual ao *gadget* apresentado na seção anterior: apenas um triângulo.

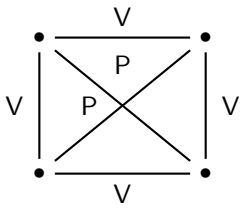
$$(x \vee \bar{y} \vee \bar{z}) \quad \Rightarrow$$

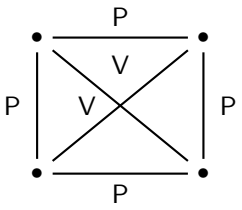


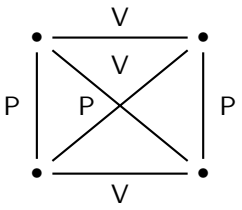
O que falta é conectar o literal das cláusulas ao literal oposto no *gadget* da variável. Mas para isso deve-se construir o *gadget* da super-aresta. E, novamente o recurso utilizado é o grafo K_4.



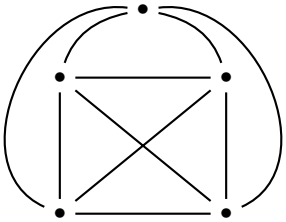
Mas diferente da redução anterior, há mais casos possíveis para coloração do grafo por meio das arestas.



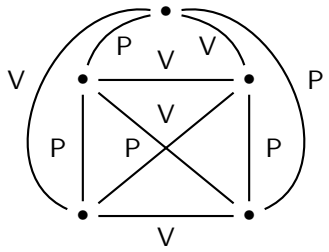




O próximo passo é inserir esse mesmo K_4 em um K_5.



E com essa nova construção não é possível completar a coloração partindo das duas primeiras colorações do grafo K_4, mas é possível completar a coloração partindo da terceira possibilidade:



Percebe-se que no K_4 dentro do K_5 há uma alternância de coloração das arestas externas (o terceiro caso possível de coloração do K_4).

Ainda é possível redesenhar esse *gadget* para o seguinte formato:

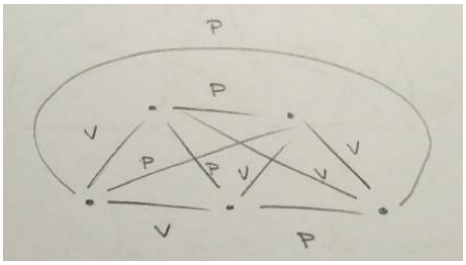


Figura 6.5: K_5 redesenhado

Com a construção nesse formato é possível visualizar na base da mesma o *gadget* da variável apresentada anteriormente, onde as arestas de cores opostas dividem um mesmo vértice.

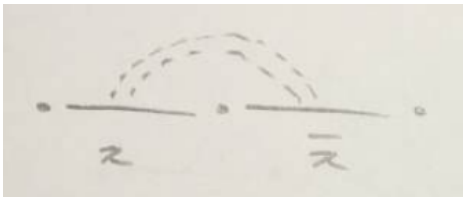
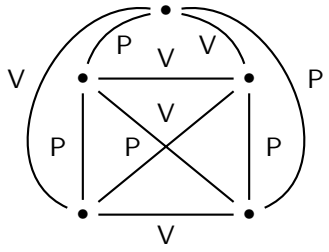


Figura 6.6: NEG - Tmon-2A

E analisando novamente o grafo K_5 inicial e visualizando as duas arestas da diagonal no centro do grafo interno K_4, temos:



É possível redesenhar o mesmo para uma segunda construção do *gadget* da super-aresta onde as duas arestas da variável não possuem vértice em comum.

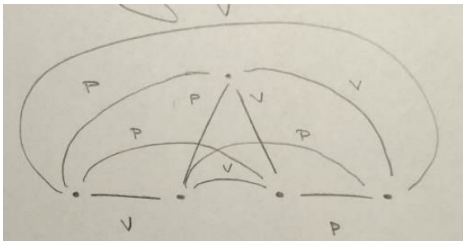


Figura 6.7: Super-aresta sem vértice compartilhado

E essa construção pode ser representada da seguinte forma:

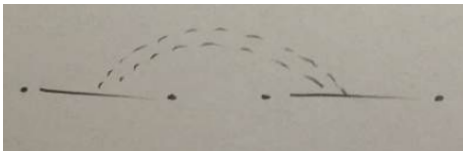


Figura 6.8: Representação 2 do NEG

Capítulo 7

Conclusão

Este trabalho iniciou com a seguinte pergunta:

- *Como é que se pensa sobre as reduções que provam a NP-completude ?*

E a resposta a obtida foram os *gadgets* lógicos. Ou seja, ao deparar-se com um problema difícil, que inicialmente desconfia-se ser um problema NP-completo, agora temos uma sequência de passos que se pode seguir:

1. identificar alguma coisa que possa fazer o papel daquilo que É ou Não é;
2. tentar estabelecer a relação de negação (NEG) entre esses elementos;
3. tentar construir um componente que relaciona esses elementos de acordo com a lógica do OU;
4. tentar modificar esse componente e utilizá-lo como base para a construção do TROU.

Esse método ainda não foi colocado à prova, porque em todos os exemplos

partiu-se de reduções conhecidas para poder chegar à uma redução baseada em *gadgets* lógicos[1].

Nesse ponto, pode-se dizer que as reduções conhecidas já estavam articulando a lógica subjacente do problema, que é explicitada pelas reduções apresentadas. Mesmo assim, não é fácil encontrar argumentos para esse tipo de hipótese.

Por outro lado, chamou a nossa atenção a semelhança que se pode notar na sequência de reduções que este trabalho apresenta.

Em parte, isso pode ser explicado pelo fato de que, após ter tido sucesso em construir as primeiras reduções, obteve-se um modelo para construir as próximas.

Mas, existem certos aspectos estruturais recorrentes que persistem em atrair a nossa atenção.

Por exemplo, o *gadget* do OU sempre envolve a articulação de uma tricotomia ou de alguma relação ternária.

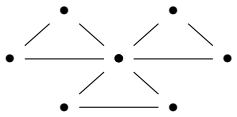
Por exemplo,

- No caso de 3Col é o vértice de 3 cores;
- No caso de CobV e CInd é o par de vértices ligado por uma aresta, acompanhado do parâmetro k

;

- No caso de PtT são os 3 triângulos no centro do componente

[1] a única exceção foi o problema TMon-2A



.

E em outros problemas, como 3DM, TMon-2V e TMon-2V, já é possível encontrar a relação ternária embutida na definição do próprio problema.

Diante disso, há diversos pares de problemas onde o primeiro é NP-completo e o segundo não é, vejamos:

- 3SAT e 2SAT;
- 3Col e 2Col;
- 3DM e bipartite matching;
- entre outros.

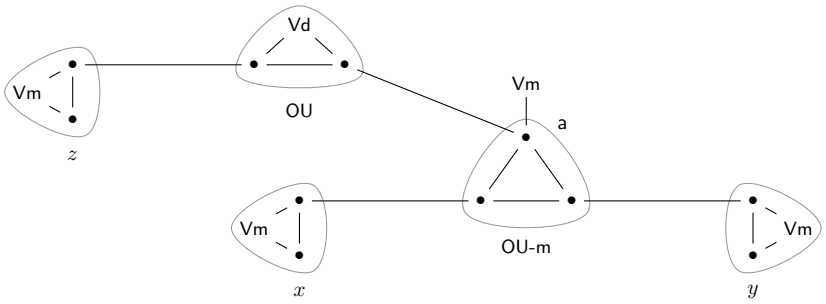
E é possível refletir se essa investigação lógica poderia jogar alguma luz sobre essa questão.

Com este trabalho ainda não se obteve muito progresso nessa direção, mas esse é um tema para trabalhos futuros.

Outro aspecto estrutural recorrente que aparece nas reduções apresentadas é a construção do TROU por meio da modificação do OU, com a introdução de um botão liga-desliga nesse componente.

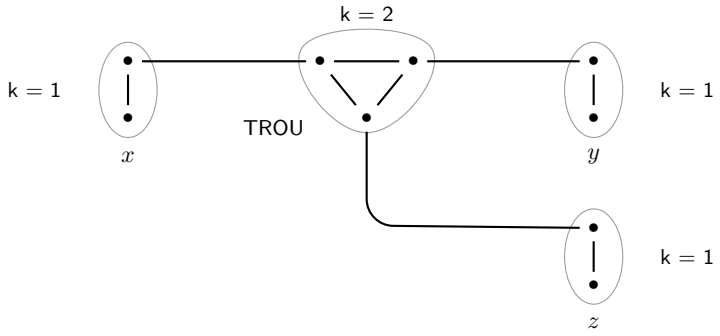
Por exemplo,

- o TROU de 3Col



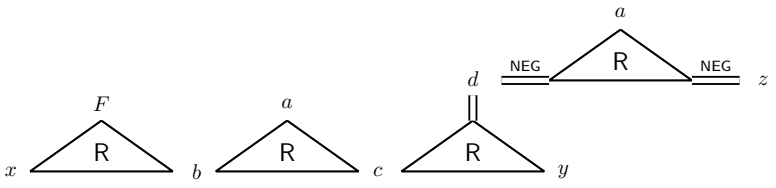
;

- o TROU de CobV



;

- o TROU de 1-em-3SAT



.

Aqui, mais uma vez, poderia-se pensar que isso é apenas um princípio heurístico de construção, que foi encontrado e adotado nesse trabalho. Mas

talvez haja algo mais. Quer dizer, considere mais uma vez a relação de negação:

$$x \quad \mathsf{NEG} \quad y$$

Se x É, *então* y Não é

Se x Não é, *então* y É

e a relação de implicação:

$$x \quad \rightarrow \quad y$$

Se x É, *então* y *também* É

Se x Não é, *então isso nada diz sobre* y

Inicialmente, a primeira relação é apresentada como um tanto rígida, no sentido de que ela vincula as duas variáveis independentemente de qualquer contexto. E foi deixado de maneira implícita a ideia de que a flexibilidade da implicação advém do fato de que as variáveis só estão vinculadas quando x se encontra no estado É — (*no outro caso, a variável y está livre para variar*).

Uma outra maneira de ver as coisas, consiste em pensar que isso é uma relação entre variáveis modulada ou regulada por um "valor verdade".

O problema é que esse é o valor verdade de uma das variáveis envolvidas na relação, o que tira um pouco de flexibilidade da situação.

Desse ponto de vista, o TROU aparece como uma solução para esse problema. Quer dizer, agora tem-se uma relação entre duas variáveis que é modulada ou regulada pelo estado de uma terceira variável.

E as nossas construções explicitariam o fato de que isso é obtido por meio da modificação da relação original, para a introdução de um ponto de regulação — (*i.e., o botão liga-desliga*).

Aqui se abre toda uma nova dimensão de investigação, pois isso conecta este trabalho com um princípio de operação muito geral, que se encontra nas redes relacionais organizacionais: o *princípio do estímulo e inibição*.

Por exemplo, é possível encontrar esse princípio nas redes químicas (metabolismo), onde um catalizador (molécula) implementa uma relação (reação entre moléculas) que pode estar ativa ou inativa, dependendo da presença ou não de um elemento (molécula) que se acopla a ele — (*em seu ponto de regulação*).

O princípio do estímulo e inibição também é a base da operação das redes neurais. E ele também é a base do funcionamento dos transistores, onde a relação entre dois sinais é modulada ou regulada por um terceiro sinal.

Agora, esse princípio é encontrado na lógica. E isso pode indicar que a linguagem humana é mais um sistema que opera de acordo com esse princípio.

Mas essa conclusão ainda é muito prematura, e necessita de mais investigação.

Finalmente, a lógica dos problemas computacionais articulada por meio dos *gadgets* lógicos é apenas o aspecto superficial que aparece após as construções destes elementos.

Mas essas construções precisam tomar alguma coisa como ponto de partida: os elementos e relações primitivas de cada problema. E aqui a diversidade reaparece outra vez. Porque os problemas são todos diferentes do ponto de vista conceitual. Mas, em todos os casos, essa diversidade serve de base para fazer a

construção da lógica — (*quando os problemas são NP-completos*)

Então, alguma coisa pode haver de comum aí.

A ideia deste trabalho é fazer essa investigação examinando as *linguagens de base* dos problemas.

E isso é mais uma coisa que pretende-se fazer no futuro.

Referências Bibliográficas

[CLR12] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Algoritmos*. Elsevier Brasil, 2012.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman and Co Ltd, 1979.

[KMTB72] Richard M. Karp, Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Reducibility among combinatorial problems. *Boston, MA: Springer US. The IBM Research Symposia Series*, 1972.

[Pap96] Christos M. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1996.

[Sip12] Michael Sipser. *Introdução à Teoria da Computação*. Cengage Learning, 2012.