Capítulo 2: Programação com Restrições

Planeamento

"Programação" significa, neste capítulo, "planeamento", nomeadamente o planeamento de sistemas que não evoluem com o tempo.
Neste contexto os problemas de planeamento estático caracterizam-se por um grupo de entidades quantificáveis que, genericamente, se podem colocar em três classes

- Um conjunto de **recursos** e um conjunto de **objetivos**.
  a. Tanto os recursos como os objetivos são medidos por quantidades que podem ser *discretas* ou *contínuas*.
  b. As discretas são modeladas por valores booleanos ou inteiros. As contínuas são modeladas por valores racionais ou reais.

- Um conjunto de **restrições** que relacionam recursos com objetivos.
  a. As restrições podem ser booleanas ou aritméticas. No primeiro caso exprimem-se por fórmulas proposicionais. No segundo caso exprimem-se por relações aritmética.
  b. Um problema de programação aritmética que se exprime completamente dentro da LIA diz-se se **linear**; uma tal programação é modelada por um problema PL ou PI.
     - Quando o problema de programação discreto que extravasa a capacidade descritiva da LIA designa-se por CP ("Constraint Programmig").

- Uma medida de **rendimento**, na utilização dos recursos para cumprir os objetivos, e/ou uma medida do **custo** dessa utilização.

Na sua forma mais simples, o **problema do planeamento** pode-se definir como

> *a alocação de recursos a objetivos, cumprindo + restrições, maximizando o rendimento extraído desses recursos ou minimizando o seu custo.*

As lógicas geralmente usadas, neste capítulo, para descrever este problema são as que foram apresentadas no capítulo anterior: a lógica proposicional (PROP) e a lógica da aritmética linear inteira (LIA). Ambas são decidíveis e finamente axiomatisáveis e

isso permite construir ferramentas computacionais que ajudam nos problemas de planeamento.

Nomeadamente, nesse capítulo, estudamos (em fórmulas proposicionais e em formas relacionais) alguns problemas cuja solução é essencial aos problemas de planeamento: o problema da valoração de uma fórmula (EVAL) e o problema da satisfabilidade (SAT).

> Em princípio é sempre possível escrever restrições usando lógicas mais expressivas. No entanto a decidibilidade (ou não-decidibilidade) da maioria de tais lógicas torna pouco prático o seu uso a menos que esse uso se restrinja a certos fragmentos decidíveis. Nomeadamente a resolução do problema SAT obriga ao uso de fragmentos que, na prática, pouco se distinguem das duas lógicas aqui referidas.

No entanto os problemas "Constraint Programming" (CP) exigem extensões da LIA que **não** são decidíveis. A resolução destes problemas depende da capacidade do "solver" em resolver alguma instâncias do problemas SAT com essas extensões.

> O uso da package `or-tools` em Python tem suporte distintos para CP e para problemas lineares.
> Tem também suporte para algumas formas de problemas não-lineares usando variáveis contínuas.

O problema do planeamento vai além dos algoritmos básicos EVAL e SAT: o problema tem ainda uma componente de **optimização** , a escolha de uma solução que é a "melhor" (seguindo um determinado critério) de entre todas as possíveis soluções do SAT.

Genericamente a resolução desses problemas actua a vários nivéis:

1. Ao nível da **verificação** pretende-se, dado um candidato a solução x, provar que ele satisfaz todas as restrições e, por isso, é uma solução viável.
   A verificação inclui a análise da consistência das restrições (e por isso envolve o uso do SAT).
   A prova pode-se resumir à execução da computação $\mathrm{EVAL}(x)$ mas também pode requerer uma "testemunha" ω que possoa ser verificada numa computação $\mathrm{EVAL}(x, \omega)$.

2. Ao nível da **alocação**, quando é possível satisfazer todas as restrições, procede-se à alocação dos recursos a compromissos.
   Quando não é possível satisfazer todas as restrições, analisa-se (eventualmente) como algumas das restrições podem ser relaxadas. Esta é uma fase iterativa que age "fora da lógica" e ao nível do ambiente de suporte às ferramentas lógicas.

3. Na fase de **optimização**, o conjunto de "restrições razoáveis" já foi definido e sabe-se que existem soluções viáveis que satifazem todas as restrições. Agora o objetivo é a escolha da "melhor solução": a que maximiza o rendimento ou minimiza o custo. Para isso quase todos os algoritmos de SAT têm uma variante que realiza essa optimização.

Um problema de **optimização linear** usa restrições lineares e um critério de otimização que é um termo LIA; em caso contrário a optimização é **não-linear**. Quando o problema é discreto designa-se genericamente por **CP** ("Constraint Programming").

Uma outra forma de problema de optimização não-linear, designado por QP ("Quadratic Programming"), é um problema contínuo onde as restrições são lineares mas o critério de optimização é quadrático (i.e. um polinómio do 2º grau).

Os diferentes tipos de problemas de planeamente estático são caracterizados, essencialmente, pelo número e tipo de restrições usadas (linear ou CP) e pelo tipo dos critérios de optimabilidade (nenhum, linear ou quadrático). Nas diferentes secções deste capítulo vamos apresentar alguns tipos básicos destes problemas.

Problemas de Alocação Linear

Nos problemas de alocação linear as restrições organizam-se em dois grupos: **limitações**, que exprimem limites aos recursos usados, e **obrigações**, que exprimem mínimos a cumprir para satisfazer os objetivos.
Estes dois tipos de restrições estão relacionadas com as duas formas possíveis de optimização: a orientada à minimização de custos pode ser expressa como uma

limitação enquanto que a orientada ao rendimento pode ser expresso como uma obrigação.

Nestes problemas as funções de optimização (rendimento ou custos) são lineares e todas as restrições se escrevem como *relações lineares.* Nomeadamente

- Uma função linear nas variáveis $x \equiv (x_0, \cdots, x_{n-1})$ é da forma $\phi_c(x) = \sum_{i<n} c_i x_i$ sendo $c \equiv (c_0, \cdots, c_{n-1})$ um vetor de constantes.
- A função ϕ_c é *positiva* (ou *convexa*) se todos os c_i são positivos. As funções rendimento e custo são sempre lineares positivas.
- Uma limitação é uma relação da forma $\phi_c(x) \leq c_n$ sendo ϕ_c uma função linear positiva. Tais relações dizem-se **convexas** porque o seu domínio é uma região convexa do espaço.
- Uma obrigação é uma relação $\phi_c(x) > c_n$ sendo ϕ_c uma função linear positiva.
- Tais relações dizem-se **côncavas** porque o seu domínio é o complemento de uma região convexa do espaço.

Sem perda de generalidade nestes problemas as variáveis x_i só tomam valores positivos. Se o domínio de todos os x_i está contido nos inteiros positivos N, então o modelo diz-se *discreto* e é resolvido com PI. Se alguma das variáveis tem um domínio nos reais positivos R_+ o problema só é resolúvel com MIP.
Como caso particular de problema discreto temos a situação em que todas as variáveis x_i têm domínio $\{0, 1\} \subset \mathrm{N}$. Nesse caso o problema é **binário** ou **booleano.**

Vamos considerar, a título de exemplo, alguns problemas lineares nesta classe.

Alocação de Operadores a Máquinas

Neste problema os recursos são N operadores (enumerados $1..N$) e os objetivos, enumerados de $1..M$, são a operação de M máquinas. Supõe-se que este problema é completamente estático no sentido que a alocação não depende do tempo.

Uma alocação que dependa do tempo é um horário como veremos no exemplo seguinte.

As variáveis do problema tomam valores binários $\{0,1\}$ interpretados como inteiros.

Existe uma variável $X_{i,j}$ para cada combinação de operador $i = 1..N$ e máquina $j = 1..M$; as variáveis são agrupadas numa matriz de variáveis e têm a seguinte interpretação

$X_{i,j} = 1$ se e só se o operador i está alocado à máquina j

Estas matrizes designam-se por **matrizes de alocação** e são típicas de todos estes problemas.

Pode-se estender este modelo considerando que as variáveis tomam valores reais no intervalo [0,1]. Neste caso $X_{i,j}$ representa a **probabilidade** de o operador i estar alocado à máquina j .
Esta é uma situação onde a alocação é **estocástica** : não existe a certeza absoluta de qual é a máquina a que o operador está associado mas apenas existe uma relação entre probabilidades de tal alocação ocorrer. Calculadas estas probabilidades então uma alocação concreta realiza-se gerando aleatoriamente valores binários com distribuições estatísticas guiadas por essas probabilidades.

As "obrigações" exprimem sempre que "algo tem de acontecer"; genericamente, em qualquer sistema (estático ou dinâmico) tais asserções designam-se por **propriedades de animação**.

As "limitações" exprimem sempre que "algo não pode acontecer"; genericamente estas asserções são designadas por **propriedades de segurança**.

Nos problemas de alocação as propriedades de animação são descritas por relações côncavas enquanto que as propriedades de segurança são descritas por relações convexas.

Para exprimir que, para qualquer máquina j , para a operar são necessários um mínimo de $\mathbf{t}_j$ operadores, usamos um conjunto de obrigações,

$$\sum_{i=1}^{N} X_{i,j} \geq \mathbf{t}_j \quad \text{para todo } j = 1..M$$

Este problema tem, pelo menos, duas classes de limitações. Para exprimir que operador i pode estar alocado a não mais do que $\mathbf{m}_i$ máquinas usa-se

$$\sum_{j=1}^{M} X_{i,j} \leq \mathbf{m}_i \quad \text{para todo } i = 1..N$$

Adicionalmente é preciso exprimir a restrição de que um operador só pode ser alocada a uma máquina se estiver habilitado para operar essa máquina. A habilitação dos vários operadores é especificada por uma matriz constante $\mathbf{H} \in \{0,1\}^{N\times M}$ com a interpretação

$\mathbf{H}_{i,j} = 1$ sse o i-ésimo operador está habilitado para operar a j-ésima máquina

A restição é agora

$$X_{i,j} \leq \mathbf{H}_{i,j} \quad \text{para todo } i = 1..N \text{ e todo } j = 1..M$$

Como critério de optimabilidade pode-se maximizar o número de máquinas que estão a ser operadas ou minimizar o número de operadores que estão ativos. Para incluir qualquer destes critérios de optimabilidade é necessário fazer algumas modificações na formulação das restrições e no conjuntos de variáveis.

a. *Maximização do rendimento*

   Vamos adicionar novas variáveis booleanas Y_j, uma por cada máquina, com a interpretação: $Y_j = 1$ sse a j-ésima máquina está a operar.

   Agora a única obrigação vai ser substituida por

   $$\sum_{i=1}^{N} X_{i,j} \geq \mathbf{t}_j Y_j \quad \text{para todo } j = 1..M$$

   Agora o número total de máquinas a operar é obviamente a soma de todos os Y_j; assim o rendimento a maximizar é dado por

   $$\mathit{rendimento} \equiv \sum_{j=1}^{M} Y_j$$

b. *Minimização do custo*

   A estratégia é semelhante: acrescentar novas variáveis e alterar algumas restrições. Neste caso vamos adicionar novas variáveis booleanas Z_i, uma por cada um dos operadores, com a interpretação: $Z_i = 1$ se e só se o i-ésimo operador está ativo.

   A restrição que deve ser alterada é a que impõe um limite no número total de máquinas que um operador pode operar; esse limite passa a ser

   $$\sum_{j=1}^{M} X_{i,j} \leq \mathbf{m}_i Z_i \quad \text{para todo } i = 1..N$$

   Finalmente o custo é o número total de operadores ativos que é dado pelo soma dos Z_i's.

   $$\mathit{custo} \equiv \sum_{i=1}^{N} Z_i$$

Uma terceira alternativa seria construir uma nova função a maximizar que combine num só critério ambos os objetivos anteriores. Basta fazer

$$objetivo \equiv rendimento - custo$$

Maximizar esta diferença vai balancear, na solução, os efeitos da maximização de rendimento com os da minimização de custos.

Finalmente uma combinação de critérios mais geral usa pesos diferentes para cada operador e cada máquina, traduzindo assim rendimentos diferente às várias máquinas e custos diferentes aos vários operadores. Pode-se definir um objetivo generalizado

$$obj \equiv \sum_{j=1}^{M} r_j Y_j - \sum_{i=1}^{N} c_i Z_i$$

em que os r_j e os c_i são pesos positivos.

Este exemplo, apesar de muito simples, tem a maioria das componentes que ocorrem em qualquer problema de alocação. Alguns aspetos que ocorrem em todos são

1. O uso de variáveis booleanas binárias para representar alocações mas também "flags" que indicam quando certos recursos ou objetivos estão ativos.
2. Restrições do tipo *obrigação* representadas por relações côncavas;
3. Restrições do tipo *limitação* representadas por relações convexas;
4. Os critérios de optimabilidade são sempre da forma da maximização de rendimento e/ou minimização de custos em critérios como indicados na nota acima.

Construção de horários

A formulação da alocação de operadores a máquinas, que vimos atrás, é muito simplificada e por isso um pouco afastada de casos reais. A formulação do horário escolar, que vamos apresentar em seguida, é mais complexa e também mais próxima dos problemas reais de construção de horários.

O problema é caracterizado por recursos, objetivos e restrições específicos do tipo de horário. A título de exemplo vamos aqui caracterizar um horário escolar para uma semana de atividades.

Um horário escolar deste tipo lida com várias entidades; nomeadamente

- *Salas* e *"slots"* de tempo; cada slot é determinado por um dia da semana e uma hora do dia.
- *Disciplinas* e *sessões*: cada disciplina tem várias sessões letivas ao longo da semana.
- *Docente.*

Neste problema de alocação "salas", "slots" e "docentes" são recursos; "disciplinas" e "sessões" são compromissos. Por isso a dimensão total da matriz de alocação é

$$\text{num salas} \times \text{num.slots} \times \text{num.docentes} \times \text{num.disciplinas} \times \text{num.sessões}$$

Mesmo que realisticamente o número de entes de cada um destes tipos não seja muito grande, o produto acaba por ser grande: a dimensão da matriz de alocação é da ordem das dezenas ou centenas de milhar.

As restrições do tipo obrigação indicam que "algo tem de acontecer"; por exemplo

1. Toda a $(\text{disciplina}, \text{sessão})$ tem alocado um e só um $(\text{docente}, \text{sala}, \text{slot})$
2. Toda a disciplina tem alocado um número de slots definido pela sua escolaridade.

As restrições do tipo limite indicam que "algo não pode acontecer"; por exemplo

1. O número de "slots" alocados a cada docente não ultrapassa um número fixado pelo seu contrato.
2. Um docente não pode ser alocado a uma disciplina para a qual não está habilitado.
3. Um docente não pode ser alocado a um "slot" se não estiver disponível nesse "slot".
4. Todo o par $(\text{sala}, \text{slot})$ não pode ser alocada a dois pares distintos $(\text{disciplina}, \text{sessão})$.
5. etc.

O critério de optimabilidade segue a abordagem usada em todos os problemas de alocação: maximizar um rendimento (relacionado com o cumprimento dos objetivos) e/ou minimizar um custo (relacionado com o uso dos recursos).

Problemas de Incidência: empacotamente, cobertura, partição e seus duais.

Problemas de incidência lidam essencialmente com domínios finitos, "pontos" desses domínios, subconjuntos desse domínio e com a relação de "pertença" $w \in S$.
Os dados básicos em todos estes problemas são:

- Um conjunto finito $W \subseteq \mathrm{N}$ designado por **universo**.
  - O domínio de todos os subconjuntos de W, o "power-set" de W, representa-se por $\wp(W)$ ou por $\mathbf{2}^W$. É também um conjunto finito.
  - Para cada $w \in W$ (os "pontos") representa-se por $\hat{w}$ (a "cobertura" de w) o conjunto de todos os subconjuntos de W que contêm w;
    $$\hat{w} \equiv \{S \subseteq W) \mid w \in S\}.$$
  - Equivalentemente, para todos $x \in W$ e $S \in \wp(W)$, verifica-se
    $$w \in S \text{ sse } S \in \hat{w}.$$
- Uma enumeração de elementos do universo W
  $$E \equiv \{w_1, w_2, \cdots, w_N\} \quad \text{com } w_i \in W \text{ para } i = 1..N$$
- Uma enumeração de elementos do "powerset" $\wp(W)$
  $$F \equiv \{S_1, S_2, \cdots, S_M\} \quad \text{com } S_j \in \wp(W) \text{ para } j = 1..M$$

- Uma **matriz de incidência** $\mathrm{A} \in \{0,1\}^{N \times M}$ que verifica
  $$\mathrm{A}_{i,j} = 1 \quad \text{se e só se} \quad w_i \in S_j \quad \text{se e só se} \quad S_j \in \hat{w}_i$$
  para todos $i = 1..N$, $j = 1..M$

Nos problemas de incidência toda a informação sobre os pontos w_i e os conjuntos S_j abstrai-se na matriz de incidência A. Nomeadamente não são relevantes os pontos que não pertencem à enumeração E nem os conjuntos que não pertencem à enumeração F.

Os problemas de incidência procuram determinar algum conjunto X de pontos w_i que verifica uma determinada condição ou então procura determinar um conjunto Y de subconjuntos S_j que também verificam outra condição.
Para representar essas incógnitas usamos vetores de variáveis:

- O vetor X, que descreve um conjunto $X \subseteq E$ é instanciado com valores $\{0,1\}^N$ e é interpretado como
  $$X_i = 1 \quad \text{se e só se} \quad w_i \in X$$

- O vetor Y, que descreve um conjunto $Y \subseteq F$, é instanciado com valores $\{0,1\}^M$ e é interpretado como

$$Y_j = 1 \quad \text{se e só se} \quad S_j \in Y$$

Para ambos os casos o tamanho dos conjuntos é dado pela soma das componentes do vetores.

$$|X| \equiv \sum_{i=1}^{N} X_i \quad \text{e} \quad |Y| \equiv \sum_{j=1}^{M} Y_j$$

Com estas interpretações, dos conjuntos X, Y e da matriz de incidência A, conclui-se que:

i. $\mathrm{A}_{i,j} \cdot X_i = 1$ se e só se $w_i \in S_j \cap X$ se e só se $w_i \in S_j$ e $w_i \in X$.

ii. $\mathrm{A}_{i,j} \cdot Y_j = 1$ se e só se $S_j \in \hat{w}_i \cap Y$ se e só se $w_i \in S_j$ e $S_j \in Y$.

Os problemas de incidência são

Problemas do empacotamente e co-empacotamento

Estes problemas são também designado por **"point packing" e "set packing"**, respetivamente**.**

Point Packing

> *Determinar o maior conjunto $X \subseteq E$ que não intersecta qualquer $S_j \in F$ em mais do que um elemento.*

Atendendo à observação 1. anterior, para cada conjunto $S_j \in F$, o número de vezes que algum $w_i \in X$ pertence a S_j é dado por $\sum_{i=1}^{N} \mathrm{A}_{i,j} \cdot X_i$.

Por isso o problema é caracterizado por M limitações (uma por cada elemento de F),

$$\sum_{i=1}^{N} \mathrm{A}_{i,j} \cdot X_i \leq 1 \quad \text{para todo} \, j = 1, \cdots, M$$

e, no contexto da programação inteira binária, o critério de optimabilidade é a maximização do tamanho do conjunto X

$$\text{maximize} \sum_i X_i$$

Set Packing

> *Determinar o maior conjunto* $Y \subseteq F$ *que não intersecta qualquer* $\hat{w}_i$ *em não mais de um* S_j *. Equivalentemente, todo o* w_i *está contido em quanto muito um* $S_j \in Y$.

Da observação 2. anterior e por analogia com o problema de "point packing" o problema é caracterizado por N limitações (uma por elemento de E)

$$\sum_{j=1}^{M} \mathrm{A}_{i,j} \cdot Y_j \leq 1 \quad \text{para todo } i = 1, \cdots, N$$

e o critério de optimabiliade assenta na maximização do tamanho de Y

$$\text{maximize } \sum_j Y_j$$

É óbvio que o problema "set packing" é formulado exatamente do memo modo que o "point packing" mas para a matriz transposta $\mathrm{A}^\top$.

Problemas de cobertura e co-cobertura.
Estes problemas são também conhecidos por **"point cover"** e **"set cover"**, respetivamente.

Point Cover

> *Determinar o menor conjunto* $X \subseteq E$ *que intersecta todos os* S_j *.*

Agora as restrições são obrigações, uma por cada conjunto S_j : todos têm de intersectar X pelo menos uma vez. Daí as restrições se escreverem

$$\sum_{i=1}^{N} \mathrm{A}_{i,j} \cdot X_i \geq 1 \quad \text{para todo } j = 1, \cdots, M$$

e o critério de optimabilidade é a minimização do tamanho de X.

$$\text{minimize } \sum_i X_i$$

Set Cover

> *Determinar o menor* $Y \subseteq F$ *tal que todo* w_i *está contido em pelo menos um elemento* $S_j \in Y$.

Mais uma vez as restrições são obrigações, uma por cada elemento w_i.

$$\sum_{j=1}^{M} \mathrm{A}_{i,j} \cdot Y_j \geq 1 \quad \text{para todo } i = 1, \cdots, N$$

e o critério de optimabiliade assenta na minimização do tamanho de Y

$$\text{minimize} \sum_j Y_j$$

Problemas da partição e co-partição ("point partition" e "set partition")
Estes problemas não têm qualquer critério de optimabilidade: pretende-se apenas encontrar uma solução.

Point partition

> *Determinar um conjunto $X \subseteq E$ que intersecta cada um dos S_j exatamente uma vez.*

As restrições são, agora, equações

$$\sum_{i=1}^{N} \mathrm{A}_{i,j} \cdot X_i = 1 \quad \text{para todo } j = 1, \cdots, M$$

e, como dissemos, não há nenhum critério de optimabilidade.

Set Partition

> *Determinar um conjunto $Y \subseteq F$ tal que todo $w_i \in E$ está contido em exatamente um conjunto $S_j \in Y$.*

Igualmente as restrições são equações,

$$\sum_{j=1}^{M} \mathrm{A}_{i,j} \cdot Y_j = 1 \quad \text{para todo } i = 1, \cdots, N$$

Problema do Portofolio

O problema do portofolio é um exemplo de optimização não-linear. Na sua versão mais simples o problema pode-se descrever pelas seguintes regras:

1. Um investidor dispõe de uma determinado montante disponível para investimento. Vamos assumir que esse montante é 1: independente da unidade monetária utilizada, todas as quantias usadas neste problema são representadas por racionais no intervalo $[0, 1]$; isto é, frações do montante total.

2. Existem n possíveis investimentos $R_1, \cdots, R_n$; cada um dos quais é modelado por uma variável aleatória.
   O valor expectável de R_i é um real positivo r_i que é um dado do problema. Seja $\overline{r}$ o vetor formado pelos r_i 's.
3. A covariância mútua do investimento R_i com o investimento R_j é a quantidade
$$\sigma_{i,j} \equiv \mathrm{E}[(R_i - r_i)(R_j - r_j)]$$
   vamos supor que os valores $\sigma_{i,j}$ são dados do problema.
4. A quantia investida em cada um destes investimentos é uma variável representada por x_i .
   Assim rendimento total do investimento será
$$R \equiv \sum_{i=1}^{n} x_i R_i$$
5. Pretende-se maximizar o valor expectável do rendimento R minimizando o risco do investimento. Modela-se o risco como a variância de R .

Para modelar este problema temos de descrever as duas quantidades que ocorrem no critério de optimização: o valor expectável do rendimento e o risco desse mesmo investimento.

O valor expectável do investimento é
$$E \equiv \mathrm{E}[R] = \sum_{i=1}^{n} x_i \mathrm{E}[R_i] = \sum_{i=1}^{n} x_i r_i$$

O risco do investimento é a variância de R calculada como
$$V \equiv \mathrm{E}[(R-E)^2]$$
Substituindo R e E pelas suas expansões e após algumas manipulações simples conclui-se
$$V = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \sigma_{i,j}$$

O objetivo de optimização é a maximização do critério
$$\mathrm{obj} \equiv E - \lambda * V$$
sendo $\lambda > 0$ um parâmetro que ajusta o peso relativo das duas componentes do critério.

Além disso existem duas restrições lineares limite

i. a soma total do investimento não pode ultrapassar o montante global 1
ii. os investimentos x_i são descritas por variáveis reais no intervalo $[0, 1]$.

O problemas pode então ser escrito como

$$\max_{x_1,\cdots,x_n} \quad (\sum_{i=1}^{n} x_i\, r_i) - \lambda\,(\sum_{i=1}^{n}\sum_{j=1}^{n} x_i\, x_j\, \sigma_{i,j})$$

sujeito a

$$\begin{cases} \sum_{i=1}^{n} x_i & \leq & 1 \\ 0 \leq x_i & \leq & 1 \end{cases}$$

Infelizmente parece que `or-tools` não suporta diretamente QP. Para alternativas consultar este link.

No entanto dado qie SCIP suporta programação quadrática é possível implementar o problema do portfólio usando o "warpper" do próprio SCIP `pySCIPopt`. O "notebook" `portfolio.ipynb` na diretoria `notebooks` ilustra o mecanismo usado.

Aplicação a Circuitos

Circuitos descrevem computações que são decomponíveis em operações simples implementadas ao nível de cada "gate". Uma descrição do circuito pode representar as operações das diferentes "gates" quer como uma restrição booleana quer ainda como uma restrição com a forma de uma relação linear inteira.

Adicionalmente um circuito tem "wires" que transportam a informação ao longo do circuito. Na descrição formal de um circuito os "wires" estabelecem as ligações entre as várias "gates" ; são representados por variáveis partilhadas pelas restrições que representam essas "gates".

Os problemas, onde este tipo de descrição por restrições é útil , estão normalmente relacionados com a **sensibilidade** do circuito a **erros** e **falhas**. Um **erro** ocorre quando o valor de um "wire" é corrompido por um evento externo; uma **falha** numa ligação ocorre quando um "wire" é removido.

A **sensibilidade** analisa a alteração das soluções de um problema (por exemplo, o EVAL ou o SAT) em um circuito afetado pelo erro ou falha.

Num circuito $n \times 1$ com m "gates" um erro é modelado, no caso mais geral, pela introdução de um vetor de erros $e \in \{0,1\}^{n+m}$ (um erro em cada "input" e cada "output" de uma "gate") que é limitado no número de componentes iguais a 1.

As falhas no circuito são modelas por um vetor $\zeta \in \{0,1\}^m$ (uma falha possível por "gate") que modifica o circuito C , removendo as "gates" i para os quais $\zeta_i = 1$ e os "wires" vizinhos.

Restrições booleanas

Os operadores que descrevem "gates" por restrições booleanas usam as conetivas proposicionais básicas, **neg**, **or** e **and,** mas também operadores gerados a partir destas três: **nand** e **xor**.

Com exceção da negação, que é uma operação unária, todas as restantes operadores neste conjunto, são binárias. Assim toda a "gate" que fosse modelada por um destes 4 operadores (**or, and, nand** e **xor)** teria necessariamente 2 "inputs".
Acresce ainda que todos estes operadores são simétricos: isto é, o resultado da operação não depende da ordem dos argumentos.

No contexto das implementações das "gates" faz sentido, por isso, estender cada um destes operadores a um qualquer número de argumentos. É esta opção seguida no sistema **Z3**: qualquer dos operadores recebe argumentos organizados numa sequência

$$\mathbf{or}(x_1, \cdots, x_n)\ ,\ \mathbf{nand}(x_1, \cdots, x_n)\ ,\ \text{etc}$$

Devido à simetria da operação binária, para o resultado da gate é irrelevante a posição na sequência; só interessa o valor do argumento e o número de vezes que ocorre.

> **Nota** O sistema **Z3** tem definidos dois operadores, **AtLeast** e **AtMost,** que avaliam, numa lista de argumentos, a cardinalidade dos que representam o valor lógico 1.

Genericamente na aritmética linear inteira é também possível codificar todos estes operadores.
Para além das conectivas boleadas standard (**or, xor, and, nand**) faz sentido codificar um operador booleano "maioria de k em n" (com $0 \leq k < n$)

$$\mathbf{maj}_{\kappa}(x_1, \cdots, x_n)$$

definido como produzindo o resultado 1 se e só o número de x_i's com o valor 1 é superior a κ.

Codificação da semântica dos operadores booleanos em LIA

#LIA

Como foi sendo indicado anteriormente, a descrição em LIA ("linear integer arithmetic") de uma restrição construída com esta família de operadores é direta. No contexto do circuito o comportamento de uma gate é definido por ums restrição igualdade

$$y = \mathbf{op}(x_1, \cdots, x_n)$$

sendo y o "wire utput" e $(x_1, \cdots, x_n)$ os "wires" dos argumentos.

Quando se usam restrições inteiras, e um sistema como o **SCIP**, para descrever uma restrição booleana

$$y = \mathbf{op}(x_1, \cdots, x_n)$$

é necessário proceder à conversão dos operadores booleanos em relações inteiras.

Na seguinte tabela apresentamos, na coluna da esquerda uma restrição booleana e, na coluna da direita, a sua equivalente na forma de um conjunto de restrições lineares inteiras

Operador	Expansão em Aritmética Linear Inteira	
$y = \mathbf{neg}(x)$	$y + x = 1$	
$y = \mathbf{xor}(x_1, \cdots, x_n)$	$y + 2w = z \quad \wedge$ $0 \leq w \leq \lfloor n/2 \rfloor$ w é uma nova variável	sendo $z = \sum_i x_i$
$y = \mathbf{or}(x_1, \cdots, x_n)$	$y \leq z \quad \wedge \quad z \leq n\,y$	sendo $z = \sum_i x_i$
$y = \mathbf{nand}(x_1, \cdots, x_n)$	$y + z \leq n \quad \wedge \quad z \geq n\,(1 - y)$	sendo $z = \sum_i x_i$
$y = \mathbf{and}(x_1, \cdots, x_n)$	$z < y + n \quad \wedge \quad z \geq n\,y$	sendo $z = \sum_i x_i$
$y = \mathbf{maj}_k(x_1, \cdots, x_n)$ para	$z \geq (k+1)y \quad \wedge$	sendo $z = \sum_i x_i$

$k < n$	$z \leq k + (n-k)y$	

A partir do par de operadores $\mathbf{neg}$, $\mathbf{maj}_k$ os outros operadores podem ser descritos na aritmética linear inteira. Por exemplo

Operador	Descrição
$y = \mathbf{min}_k(x_1, \cdots, x_n)$	$y = \mathbf{neg}(w) \ \wedge \ w = \mathbf{maj}_k(x_1, \cdots, x_n) \ \wedge \ 0 \leq w \leq 1$ w é uma *nova* variável
$y = \mathbf{or}(x_1, \cdots, x_n)$	$y = \mathbf{maj}_0(x_1, \cdots, x_n)$
$y = \mathbf{and}(x_1, \cdots, x_n)$	$y = \mathbf{maj}_{n-1}(x_1, \cdots, x_n)$

Exemplo

Problema de sensibilidade a erros

> *Dados um circuito C booleano $n \times 1$, e um vetor $z \in \{0,1\}^n$, determinar um vetor x que seja aceite pelo circuito (i.e. $C(x) = 1$) e que se distinga de z em não mais do que t posições.*

Para formalizar este problema introduz-se a variável de erro e (a determinar) e faz-se $x \equiv e \oplus z$. Em seguida reescreve-se o circuito como um conjunto de restrições que descrevem a relação

$$C(e \oplus z) = 1$$

Finalmente acrescenta-se a restrição

$$\sum_i e_i \leq t$$

Em alternativa pode-se acrescentar um critério de optimização

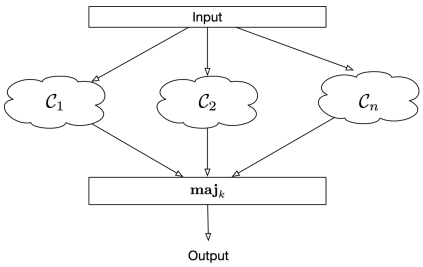
$$\text{minimize} \sum_i e_i$$

e comparar o mínimo obtido com o limite t.

Circuitos com Falhas

A grande maioria dos circuitos físicos de grande dimensão e complexidade introduzem falhas de funcionamento que derivam de certas componentes não produzirem o resultado previsível ou, simplesmente, não produzirem qualquer

resultado. Estas situações, designadas por **falhas** , são modeladas de forma probabilística através da noção de *probabilidade de falha* .



Redundância: várias componentes C_i desempenham a mesma função

Sendo as falhas inevitáveis é prudente definir estratégias para diminuir os seus efeitos. Tal é feito introduzindo **redundância**: isto é, tendo várias componentes do circuito a desempenhar a mesma função, com os mesmos "inputs", e em seguida obter um "resultado mais provável" através do "output" mais frequente nas várias componentes.

Para modelar a ocorrência de falhas num circuito pode-se usar lógica inteira ternária de valores $\{-1, 0, 1\}$; o valor -1 designa-se por "falha".

Neste modelo

- As conectivas standard (**xor , and)** são operadores n-ários que, perante um argumento "falha" dão sempre um resultado "falha". Se ambos os argumentos são positivos o resultado, com probabilidade $(1-\varepsilon)$, é dado pela tabela de verdade dos operadores ou então, com uma "probabilidade" ε, é "falha".
- O resultado da expressão $\mathbf{maj}_k(x_1, \cdots, x_n)$ é redefinido do modo seguinte: sejam k_0 o número de "inputs" $x_i \geq 0$ e k_1 o número de $x_i \geq 1$; então
  - se $k \geq k_0$ o resultado é "falha", senão
  - se $k \geq k_1$ o resultado é 0 , senão
  - o resultado é 1

Num circuito a função das "gates" $\mathbf{maj}_k$ é o de criar redundância que possa corrigir as eventuais falhas introduzidas pelas "gates" $\mathbf{xor}$ e $\mathbf{and}$.

Um exemplo comum pode-se definir em torno das seguintes hipóteses

- todas as "gates" $\mathbf{maj}_k$ implementam sempre a maioria simples; isto é, tem-se sempre $k = \lfloor n/2 \rfloor$.
- a probabilidade de falha é a mesma para todas as gates $\mathbf{or}$ e é representada pela constante ε; a probabilidade de falha das "gates" $\mathbf{and}$

é também uma constante χ; ambas as constantes fazem parte da definição do circuito.

Os "wires" de um circuito com N "inputs" e M "gates" formam uma sequência de $N+M$ variáveis w_i que tomam valores no domínio $\{-1, 0, 1\}$; as primeiras N variáveis representam "inputs" do circuito; as restantes M variáveis representam os "outputs" das várias "gates".

Cada "gate" $j \in \{1..M\}$ do circuito é representada por um tuplo

$$g_j \equiv \langle \mathrm{op}, i_0, \{i_1, \cdots, i_n\} \rangle$$

em que

- op é um "tag" no conjunto $\{\text{"xor"}, \text{"and"}, \text{"maj"}\}$ que identifica o operador da "gate"
- i_0 é o índice do "wire" output ,
- $\{i_1, \cdots, i_n\}$ é o conjunto dos índices dos "wires" inputs .

A representação do circuito está **topológicamente ordenada** se, para todas as "gates", se verifica $i_0 > i_1 > \ldots > i_n$. Assume-se sempre que esta condição se verifica.

A partir da descrição do circuito, pode-se determinar uma restrição

$$\phi_j \equiv \mathrm{R}_{\mathrm{op}}(w_{i_0}, w_{i_1}, \ldots, w_{i_n})$$

entre as variáveis w_i ("inputs" e "output") que intervêm nessa gate.

A forma da restrição é determinada pelo operador op com a codificação R_{op} que definimos atrás. Essencialmente esta codificação depende apenas do tipo de "gate" e, por isso, existem só três codificações que é necessário programar. A restrição específica de cada "gate" obtém-se aplicando uma destas codificações aos "wires" apropriados.

Note-se que cada restrição depende também das probabilidades de falha ε e χ.

O modelo do circuito é formado pelo conjunto de todas estas restrições

$$\mathrm{M} = \bigwedge_{j=1}^{M} \phi_j$$

Uma aplicação típica consiste na verificação de que, perante as probabilidades de falha ε e χ, o circuito produz um valor sem falha. Isto ocorre quando a redundância

introduzida pelas "gates" **maj** consegue anular as eventuais falhas nas "gates" **xor** e **and**.

Aplicações a Problemas de Grafos

Essencialmente um grafo G é definido por dois conjuntos enumerados (finitos ou não): um conjunto V, cujos elementos se designam por **nodos** ou **vértices**, e um conjunto E cujos elementos se designam por **ramos**, **arcos** ou **"edges"**.

$$\mathrm{G} \equiv \langle \mathrm{V}, \mathrm{E} \rangle$$

Neste curso os grafos são sempre topológicos; isto é, existe uma **topologia** que define associações entre cada ramo r e o conjunto dos seus **nodos vizinhos** e, simultaneamente, entre cada nodo a e o conjunto dos seus **ramos vizinhos**.

Por exemplo,

- um grafo é **orientado** quando a topologia associa a cada ramo r um par orientado de nodos (a, b); o primeiro elementos do par é a **origem** de r e o segundo elemento do par é o **destino** de r.
- Se o grafo for **não-orientado** a topologia associa a cada ramo r um conjunto $\{a, b\}$ com dois nodos distintos: não estão aqui identificados origens ou destinos.

A topologia determina também a noção de **ramo vizinho**: o ramo r é vizinho do nodo a se e só se a é um nodo vizinho de r.

Obviamente não estão contemplados os "loops" (ramos com um só nodo vizinho). Este tipo de grafos requer uma topologia que use "multi-sets". Existem outras variantes destas topologias que não vão ser aqui abordadas porque não são relevantes aos problemas que vamos analisar.

Muitos problemas de planeamento ocorrem em redes cuja topologia é descrita por um grafo. Quase todas são redes de comunicações ou redes de distribuição de serviços (eleticidade, água, gaz) mas ainda, numa categoria própria, temos as redes sociais.

Nestas redes a abstração essencial é a de ***ligação*** ou ***"link"***. De forma abstrata um "link" é uma via onde flui algo: esse fluxo pode ser informação (nas redes de

comunicações), pode ser energia ou um fluido (água, gaz) e pode ainda, nas redes sociais, ser *identidades*. O fluxo num "link" pode ou não ser quantificado (por números inteiros ou racionais) mas pode simplesmente ter uma interpretação booleana: existe ou não existe.

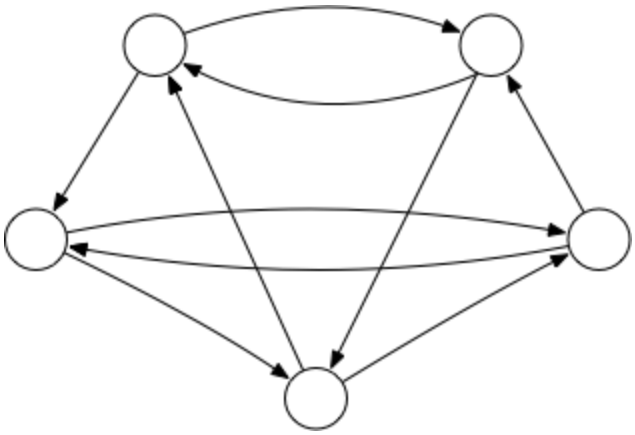
A forma como um grafo modela a topologia de uma rede segue duas abordagens:

a. **Link-as-Edge**
   Neste modo cada "link" tem identificados uma só origem e um só destino e o fluxo prossegue da origem para o destino. O grafo usa ramos para representar "links" e nodos para representar as origens e destinos desses "links".
b. **Link-as-Node**
   Agora, cada "link" não tem necessariamente identificados uma origem e um destino. No entanto existe sempre uma identificação do sentido do fluxo. Os "links" estão relacionados entre si através de relações do tipo "tem-acesso-a" ou "comunica-com". Um "link" pode estar relacionado, com a mesma relação ou com relações diferentes, com vários outros "links". O grafo representa os "links" por nodos e as relações por ramos entre esses nodos. Caso existam várias relações distintas cada um fica associada a um identificador que ocorre como marca ("label") dos ramos que a representam.

Exemplos

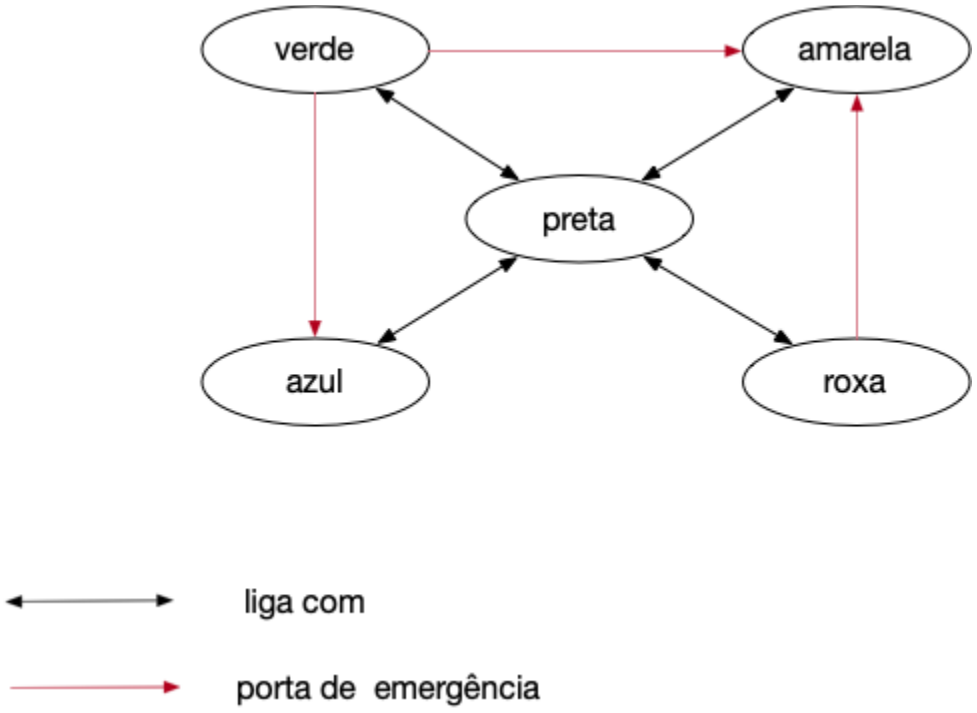
Sistemas de transporte são um exemplo frequente de redes modeladas por grafos e, nesses modelos ocorrem redes modeladas na abordagem "link-as-edge" e na abordagem "link-as-node".

Um exemplo da 1ª abordagem é uma rede de metropolitano em que os nodos representam estações e os ramos representam ligações diretas entre a origem e o destino. O grafo seguinte ilustra uma tal rede; note-se que, entre o mesmo par de estações, podem existir ligações diretas mas também caminhos que usem estações intermédias.



Rede ferroviária; modelo "linke-as-edge"

Um exemplo da 2ª abordagem a uma rede de metropolitano considera-a constituída por linhas circulares que, por simplicidade, vamos supor que têm apenas um sentido de fluxo. Como o tráfego flui permanentemente, sem origem e destinos determinados, as linhas são representadas por nodos do grafo. Neste exemplo inclui-se duas relações: a primeira, representada a negro, é a relação não orientada "liga com" ; a segunda é uma relação orientada que indica a existência de uma fuga de emergência e o sentido dessa fuga.



Rede de metro; modelo "link-as-node"

Um segundo exemplo de

Problema do Corte Mínimo

> *Dado um grafo não-orientado* $\mathrm{G} = \langle \mathrm{V}, \mathrm{E} \rangle$ *e dois nodos* $s, t \in \mathrm{V}$*, pretende-se determinar o menor conjunto de ramos* $X \subseteq \mathrm{E}$ *tal que todo o caminho* $s \rightsquigarrow t$ *contém algum ramo* $r \in X$*.*

O conjunto X designa-se por **corte mínimo** relativo ao par s, t porque se X "for cortado" do grafo (i.e. os seus ramos forem removidos) não é possivel ligar os dois nodos por um caminho.

De grafos para problema de incidência

- Pela definição de grafo, ambos os conjuntos V e E são enumerados. Sejam $n \equiv |\mathrm{V}|$ e $m \equiv |\mathrm{E}|$ o número total de nodos e o número total de ramos.

- Um subconjunto $v \subseteq \mathrm{V}$ de nodos pode ser descrito por um vetor de $v \in \{0,1\}^n$ com a interpretação: $v_i = 1$ sse o i-ésimo nodo está contido no

conjunto v.

- Igualmente cada conjunto de ramos $S \subseteq \mathrm{E}$ é descrito por um vetor de bits com a interpretação: $S_j = 1$ se e só se o j-ésimo ramo está contido no conjunto S.

- Nomeadamente todos os caminhos no grafo são determinados pelo conjunto $S \subseteq \mathrm{E}$ dos seus ramos.

A formalização do problema do corte mínimo como problema de incidência segue os passos:

1. O universo do problema é o conjunto E dos ramos do grafo.
2. Os conjuntos $S_1, \cdots, S_k$ do problema de incidência são os caminhos no grafo $s \rightsquigarrow t$.
3. A representação vetorial dos vários S_j determina uma **matriz de incidência**
$$A \in \{0,1\}^{m \times k}$$
com a interpretação
$A_{i,j} = 1$ sse o o i-ésimo ramo está contido no caminho S_j

Para interromper o caminho S_j basta remover um dos seus ramos. Assim determinar o corte mínimo X é um *problema de cobertura:*

> determinar o menor conjunto de ramos que intersecta todos os caminhos S_j.

$$\text{minimizar} \quad \sum_{i=1}^{m} X_i \quad \text{sujeito às restrições}$$
$$\sum_{i=1}^{m} A_{i,j} X_i \geq 1 \qquad \text{para todos } j = 1 \cdots k$$

Problema do Corte máximo

Nas condições do problema anterior

> *Determinar o maior conjunto de ramos X tal que existe pelo menos um caminho $s \rightsquigarrow t$ que não é cortado; i..e. mantém a conecção entre s e t.*

Neste caso a formalização será

$$\text{maximizar} \quad \sum_{i=1}^{m} X_i \quad \text{sujeito às restrições}$$
$$\sum_{i=1}^{m} A_{i,j} X_i < m \qquad \text{para todos } j = 1 \cdots k$$

uma vez que $\sum_{i=1}^{m} A_{i,j}X_i < m$ indica que pelo menos para algum i se verifica $A_{i,j}X_i = 0$: isto é, o corte X não interfere no caminho j.

Problema do Caixeiro Viajante

O **TSP** ("travelling salesman problem") é um dos problemas clássicos da teoria de grafos. Essencialmente pretende

> *Dado um grafo* $\mathrm{G} \equiv \langle \mathrm{V}, \mathrm{E} \rangle$*, não orientado e ligado, construir um caminho fechado que contenha todos os nodos do grafo sem repetições.*

Um grafo não-orientado é *ligado* quando existe sempre um caminho entre quais quer dois nodos distintos. Num tal grafo cada nodo tem, pelo menos, um ramo vizinho e cada ramo tem exatamente dois nodos vizinhos.

Alguma notação:

i. Para cada nodo $a \in \mathrm{V}$ representamos por $\delta(a)$ o conjunto dos ramos vizinhos de a.
ii. Para cada subgrafo próprio $A \subset \mathrm{G}$, representamos por $\delta(A)$ o conjunto dos ramos que não pertencem a A mas são vizinhos de algum nodo em A.

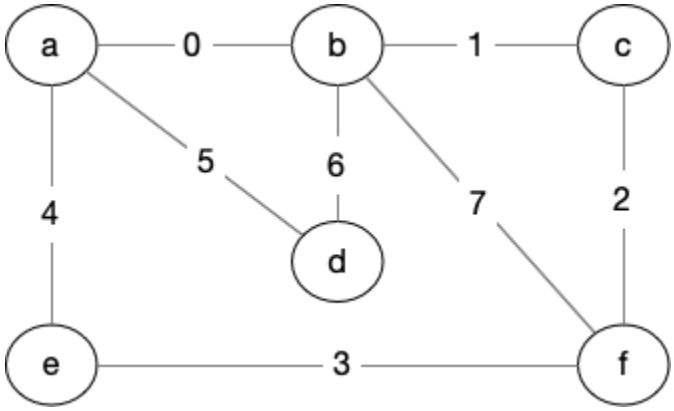
Para formalizar o TSP precisamos de carcaterizar melhor o sentido da expressão "caminho fechado". Para isso define-se

a. Um **loop** em G é um sub-grafo $L \subseteq \mathrm{G}$ que tem tantos ramos quanto nodos.
b. Um "loop" é um **ciclo** quando não contém qualquer outro "loop" distinto de si próprio.
c. Um **caminho aberto** P é um sub-grafo ligado de G em que:
   i. O número de nodos é uma unidade superior ao número de ramos,
   ii. Cada nodo a de P contém um ou dois ramos vizinhos em P.
d. Um ciclo $H \subseteq \mathrm{G}$ diz-se **Hamiltoniano** quando contém todos os nodos do grafo G sem repetições.

Nestas circunstâncias,

> *na formalização do TSP, a expressão "caminho fechado" significa "ciclo Hamiltoniano".*

Exemplo



Neste grafo o conjunto de nodos é enumerado como $\mathrm{V} \equiv \{a \cdots f\}$ e o conjunto de ramos é enumerado como $\mathrm{E} \equiv \{0 \cdots 7\}$. Um ciclo Hamiltoniano é, por exemplo, o caminho

$$H \equiv \{1, 2, 3, 4, 5, 6\}$$

Note-se que qualquer ciclo que contenha o ramo 0 ou o ramo 7 nunca é Hamiltoniano.

Exemplos da função δ neste grafo

$$\delta(a) = \{0, 4, 5\} \quad , \quad \delta(b) = \{0, 1, 6, 7\} \quad , \quad \text{etc.}$$

Seja A o sub-grafo formado pelos nodos $\mathrm{V}_A \equiv \{a, b, d\}$ e pelos ramos $\mathrm{E}_A \equiv \{0, 5, 6\}$. Então

$$\delta(A) \equiv \{1, 4, 7\}$$

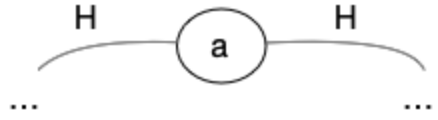
Para formalizarmos o TSB temos de ter em consideração que a incógnita H vai ser instanciada com um conjunto de ramos e, por isso, vai ser codificado num "array" com m componentes binárias interpretado como

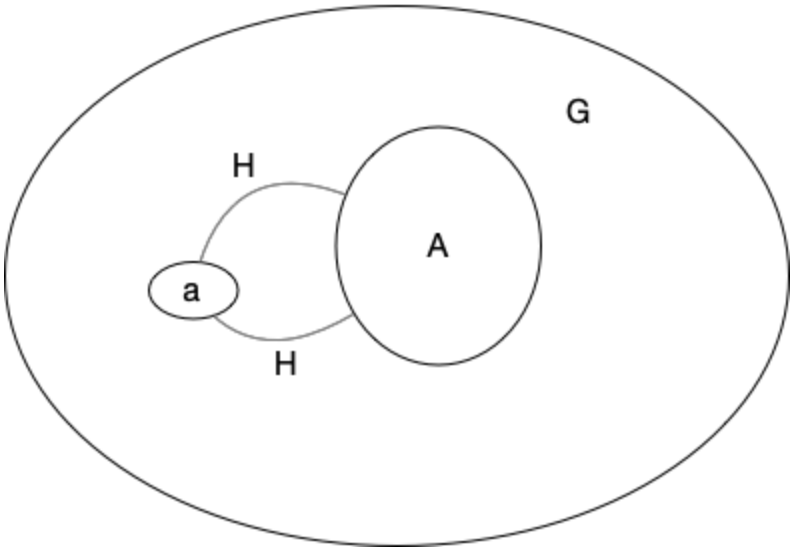
$$H_j = 1 \quad \text{sse o } j\text{-ésimo ramo de } G \text{ pertence ao caminho } H$$

As restrições do problemas são

1. Como H é um caminho cíclico, para cada nodo a nesse caminho existem exatamente dois ramos vizinhos de a que pertencem a H. Isto é o conjunto de ramos $\delta(a) \cap H$ tem exatamente dois elementos.
   Como o ciclo é hamiltoniano, todos os nodos $a \in \mathrm{V}$ pertencem a H.
   Estas restrições escrevem-se

   $\sum_{i \in \delta(a)} H_i = 2$ para todo $a \in \mathrm{V}$





2. Para assegurar que o ciclo contém todos os nodos do grafo G, temos de considerar todos os sub-grafos próprios $A \subset \mathrm{G}$.
   Para cada um desses sub-grafos tem de existir pelo menos um nodo a que não pertence a A e por onde tem de passar o ciclo H. Por isso os ramos vizinhos dos nodos de A, que não pertencem a A mas pertencem a H têm de ser pelo menos 2.

   $\sum_{i \in \delta(A)} H_i \geq 2$ para todo o subgrafo próprio $A \subset \mathrm{G}$

Como o número de restrições é muito elevado obter uma solução que satisfaza todas as restrições vai ser muito complexo. Nomeadamente é preciso percorrer todos os subconjuntos próprios dos nodos do grafo e o número desses conjuntos é exponencial com o número de nodos do grafo.

Programação Inteira Disjuntiva (DIP)

Na lógica da aritmética linear inteira, os *predicados* a n variáveis $x_1, \cdots, x_n$, são definidos por inequações com a forma genérica

$$\phi(x) \equiv c_1 \cdot x_1 + \cdots + c_n \cdot x_n \leq c_0$$

em que as constantes $c_0, c_1, \cdots, c_n$ são inteiros arbitrários (positivos ou negativos). A análise do problema da programação inteira (PI) ou da programação inteira binária (PIB) assume exclusivamente fórmulas que são uma conjunção de tais predicados.

$$\varphi \equiv \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_\kappa$$

Neste capítulo temos também insistido na "programação inteira binária" Porém muitas aplicações importantes não se podem restringir a descrições limitadas a estas fórmulas. Tais problemas exigem descrições que não só precisam de disjunções como os seus modelos não podem ser escritos apenas pelos inteiros $\{0, 1\}$: necessitam de inteiros positivos arbitrários.

A característica essencial das descrições binárias é o facto de os modelos de φ (os mundos que validam essa fórmula) são sempre em número finito. De facto pertencem ao espaço finito $\{0, 1\}^n$, em que n é o número de variáveis livres de φ. Ao invés no domínio de "programação inteira" o conjunto $\hat{\varphi} \subseteq \mathrm{N}^n$ formado pelos models de φ, o **suporte** de φ, pode não ser finito.

A caracterização do suporte de φ requer alguma terminologia:

Em primeiro lugar vamos considerar os predicados **convexos.** Recorde-se que o predicado ϕ diz-se **convexo** se todos os c_i forem não-negativos. A propriedade fundamental dos predicados convexos no contexto da programação inteira, é o facto do seu suporte ser sempre finito.

> *O suporte de um predicado convexo, com n variáveis livres, é finito e contém o vetor 0^n (o vetor com n componentes iguais a 0); se $c_0 = 0$, então 0^n é o único elemento desse suporte.*

Um predicado diz-se **côncavo** se é a negação de um predicado convexo. Portanto um predicado côncavo, tem sempre suporte infinito e esse suporte não contém o vetor 0^n.
Adicionalmente, se $c_0 = 0$ o predicado côncavo tem suporte $\mathrm{N}^n \setminus 0^n$.

Exemplo
Considere-se a inequação

$$2x_1 + x_2 \leq 3$$

cuja representação "default" é o vetor $(3, 2, 1)$.
A inequação define um predicado convexo uma vez que todos os seus coeficientes são positivos. O seu suporte é

$$\{(0,0),\ (0,1),\ (0,2),\ (0,3),\ (1,0),\ (1,1)\}$$

A negação desta inequação é

$$2x_1 + x_2 > 3$$

equivalente a $-2x_1 - x_2 \leq -4$; a sua representação é o vetor $(-4, -2, -1)$.
Esta inequação determina, por definição, um predicado côncavo. Note-se que o suporte é infinito mas não contém a origem $(0, 0)$.

Vamos agora considerar uma fórmula conjuntiva $\varphi \equiv \phi_1 \wedge \cdots \wedge \phi_\kappa$. Esta fórmula diz-se **convexa** quando <u>todos</u> os ϕ_j são convexos; diz-se **fracamente convexa** quando <u>um</u> dos ϕ_j é convexo.

Exemplo
Se ϕ for convexo e $\varphi \equiv \phi \wedge \neg\phi$, então vemos que φ é fracamente convexo (porque contém ϕ) mas o seu suporte é vazio; isto é, a fórmula é inconsistente.

O suporte de φ é a interseção dos suporte dos vários ϕ_j (i.e. $\hat{\varphi} \equiv \bigcap_j \hat{\phi}_j$) dado que w só valida φ se validar todos os ϕ_j. Basta que um do $\hat{\phi}_j$ seja finito para que $\hat{\varphi}$ também seja finito. Logo

> *O suporte de uma fórmula fracamente convexa é finito e, ou é vazio ou então contém 0^n.*

Recorde-se que uma fórmula conjuntiva $\varphi \equiv \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_\kappa$ pode ser descrita por uma matriz dos coeficientes dos seus predicados ϕ_i. De fato

- Seja $x = (x_1, x_2, \cdots, x_n)$ o vetor das variáveis com valores em N_+.
- Cada ϕ_j é determinado por um vetor $\overline{c}_j \in Z^n$ formado pelos coeficientes
  $$\overline{c}_j \equiv (c_{j,1}, \cdots, c_{j,n}) \quad \text{com} j = 1 \cdots \kappa$$
  e por um termo independente $c_{j,0}$.
  Seja C a matriz de elementos $c_{j,i}$ e seja $c_0 = (c_{1,0}, \cdots, c_{\kappa,0})$ o vetor dos termos independentes.
- A representação de φ é a matriz $C \in Z^{\kappa \times n}$ cuja j-ésima linha é o vetor $\overline{c}_j$ e pode-se escrever a conjunto de restrições φ na forma matricial como
  $$C\,x \leq c_0$$

Por motivos que serão evidentes em breve é conveniente introduzir uma variável binária y e escrever as restrições como um conjunto de relações

$$\begin{cases} C\,x & \leq & c_0\,y \\ y & = & 1 \\ x_i, y & \geq & 0 \quad \text{para} \quad i = 1..n \end{cases}$$

O modelo conjuntivo φ é agora dado pela relação nas variáveis x, y

$$\varphi(x,y) \equiv (C\,x \leq c_0\,y) \wedge (x, y \geq 0)$$

A restrição $(y = 1)$ tem um papel especial e vai ser absorvida numa restrição mais geral como veremos em seguida.

A modelação mais genérica de problemas através da programação inteira usa fórmulas construídas como a disjunção de um número finito de fórmulas conjuntivas

$$\eta \equiv \varphi^{(1)} \vee \varphi^{(2)} \vee \cdots \varphi^{(m)}$$

em que cada $\varphi^{(k)}$ é uma conjunção de predicados determinada por uma matriz $C^{(k)}$ e todos têm o mesmo conjunto de variáveis livres $x_1, x_2, \cdots, x_n$ e tem-se ainda uma variável y

$$\varphi^{(k)}(x,y) \equiv (C^{(k)}\,x \leq c_0^{(k)}\,y) \wedge (x, y \geq 0)$$

A abordagem que vamos apresentar exige que cada um dos $\varphi^{(k)}$ seja fracamente convexo. Isto é, cada matriz $C^{(k)}$ tem pelo menos uma linha em que os elementos são todos positivos.

Como habitualmente o problema da programação inteira pretende verificar se η é inconsistente e, se não for, apresentar um modelo x de η. Isto é, um vetor $x \in \mathrm{N}_+^n$ e um inteiro $y \in \{0, 1\}$ tais que,

$$\exists\ k \in \{1..m\}\ \cdot\ x, y \models \varphi^{(k)}$$

Para formalizar este problemas vai-se resolver individualmente o problema para cada um dos $\varphi^{(k)}$, modificando ligeiramente cada predicado, e construir um problema conjuntivo a partir dessas reformulações.

1. Constrói-se uma cópia ("clone") $x^{(k)}, y^{(k)}$ para o tuplo x, y de variáveis usados na resolução do problema simples. Isto é expande-se o vector de variáveis (x, y) em m cópias $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, $\cdots$, $(x^{(m)}, y^{(m)})$

2. Converte-se as diferentes alternativas em restrições conjuntivas aplicadas a cada par de variáveis $(x^{(k)}, y^{(k)})$. Isto é, constrói-se as restrições
$$\varphi_k^{'} \equiv (C^{(k)} x^{(k)} \leq c_0^{(k)} y^{(k)}) \wedge (x^{(k)}, y^{(k)} \geq 0)$$
para cada $k = 1, .., m$.
3. Acrescenta-se uma restrição adicional que estabelece a relação entre as várias alternativas
$$\mu \equiv (\textstyle\sum_{k=1}^{m} y^{(k)} = 1)$$

4. O modelo global do problema nas $m \times (n + 1)$ variáveis é agora dado pela conjunção
$$\mathrm{M} \equiv \quad \mu \wedge \bigwedge_{k=1}^{m} \varphi_k^{'}$$

Se existir uma solução $\{(\overline{x}^{(k)}, \overline{y}^{(k)})\}_{k=1}^{m}$ para a verificação de M então o modelo $\overline{x}$ do problema original calcula-se como

$$\overline{x} = \textstyle\sum_k \overline{x}_{(k)}$$

> *Note-se que a restrição* $\sum_{k=1}^{m} y^{(k)} = 1$ *força a que um e só um dos* $y^{(k)}$ *seja igual a* 1 *; os restantes são todos iguais a* 0.

> *No caso em que* $y^{(k)} = 1$, *a fórmula modificada* φ'_k *coincide com a fórmula original* $\varphi^{(k)}$ *; logo a testemunha* $x^{(k)}$ *coincide com uma testemunha da validade de* $\varphi^{(k)}$.
>
> *Nos restantes casos, em que* $y^{(k)} = 0$, *o vetor* $x^{(k)} \equiv 0^n$ *é o único que é um modelo de* φ'_k.
>
> *Por isso a soma* $\sum_k \overline{x}^{(k)}$ *constrói a solução do problema original.*

Problemas do k-centros num espaço de vetores de *bits*.

Os **problemas dos k-centros** formam uma questão clássica em *teoria da aprendizagem*. Como todos os problemas nesta categoria, o ponto de partida é um conjunto $E = \{e_1, e_2, \cdots, e_M\}$ finito de **experiências** (aka **exemplos**)
Em geral o objetivo da aprendizagem é *extrair* da informação "em bruto" E algum *padrão unificador* ; i.e. uma regra simples que descreva uma (ou mais) característica comum a todas as experiências.

> Daqui resulta a designação alternativa de "*data mining*" para a teoria da aprendizagem.

Muitos problemas de aprendizagem envolvem *distâncias* entre pontos e, por isso, formalizam-se no contexto de um **espaço métrico** (M, d). Recorde-se que a *métrica* (ou *distânca*) entre dois pontos $d(x, y)$ é um valor real positivo tal que, para todos os pontos $x, y, z \in \mathrm{M}$

i. $d(x, y) = 0$ sse $x = y$
ii. $d(x, y) = d(y, x)$
iii. $d(x, y) \leq d(x, z) + d(z, y)$

Os problemas dos k-centros desenvolvem-se no contexto de um espaço métrico: os exemplos $e \in E$ são pontos em M. O problema tem como objetivo calcular (ou verificar a existência) de k pontos de M, designados por **centros**, tal que nenhuma

experiência $e \in E$ dista de um desses centros, que depende de e, mais do que uma dada tolerância $t > 0$.

Para ajudar a clarificar esta ideia vamos enumerar todas as experiências e todos os centros,

$$E \equiv \{e_1, e_2, \cdots, e_M\} \quad \text{e} \quad A \equiv \{a_1, a_2, \cdots a_K\}$$

- As experiências E são dados do problema enquanto que os centros A são incógnitas.
- Parâmetros do problema são as dimensões M, K e a tolerância t.
- Uma incógnita auxiliar: uma matrix binária $X \in \{0,1\}^{K \times M}$ com a interpretação

  $X_{i,j} = 1$ se e só se a_i é o centro mais próximo da amostra e_j

A matriz X define os "clusters" de experiências associadas a cada centro; i.e., para cada a_i, determina o conjunto $\{ e_j \mid X_{i,j} = 1 \}$.

Estas entidades devem satisfazer algumas restrições:

a. *Para cada e_j, existe um único centro a_i que é <u>o mais próximo</u> de e_j.*

$$\forall j \in 1..M \cdot \quad \sum_{i \in 1..k} X_{i,j} = 1$$

b. *Para cada e_j, a sua distância ao "centro mais próximo" não excede t.*

$$\forall j \in 1..M \cdot \quad \sum_{i \in 1..k} X_{i,j} \cdot d(a_i, e_j) \leq t$$

Existem algumas semelhanças entre estas restrições e as usadas nos problemas de alocação. Nomeadamente ambas usam uma matriz boleada de alocação X, em que o elemento $X_{i,j}$ representa a alocação de um recurso (neste caso, o centro) a um compromisso (neste caso, a experiência).

Em problemas de alocação típicos, como os horários, cada elemento $X_{i,j}$ é multiplicado por uma constante que representa o custo dessa alocação. Assim o sistema de inequações resultante é linear.

Porém, no problema dos K-centros o custo é a distância $d(a_i, e_j)$ que vai depender do valor da variável a_i. Assim o termo $X_{i,j} \cdot d(a_i, e_j)$ contém multiplicações de variáveis.

Por isso, nesta formalização do problema de k-centros temos um problema de programação inteira não-linear.

Vamos ver que na melhor das hipóteses (a única onde estas abordagem tem solução) a distância $d(a_i, e_j)$ vai ser escrita como uma expressão linear em a_i. Por isso os termos $X_{i,j} \cdot d(a_i, e_j)$ serão quadráticos. Nestes caso resolver um problema de k-centros envolve programação inteira quadrática. Veremos também que e possível construir uma versão restrita deste problema que pode ser codificado em programação inteira linear.

Neste contexto faz sentido definir duas formas possíveis para os problemas dos K-centros:

A. *Como um problema de satisfabilidade.*
   Dados os "inputs" E, t verificar se existem centros A e alocações X que satisfazem as restrições acima.
   Este é um problema típico de verificação que se resolve definindo variáveis A e X associadas aos "outputs" do problema e usando um algoritmo de SAT no conjunto de restrições desse problema.
   O algoritmo de SAT, se as restrições forem satisfazíeis, instancia as variáveis com uma solução do problema.
B. *Com um problema de minimização.*
   Neste problema t deixa de ser um "input": é uma variável cujo valor é minimizado. Procura-se determinar o valor mínimo de t para o qual para o qual as restrições são satisfazíveis. O único "input" é o *dataset* E para além, óbviamente, das constantes M, K.

Em alternativa poder-se-ia considerar t como "input" e procurar minimizar o valor de K; afinal parece óbvio que quanto menores forem t ou K mais difícil é satisfazer as restrições. Porém isto envolveria ter K como variável que aparece, nas restrições, como limite dos somatórios; o grau de não-linearidade envolvido não permitiria formalizar este problema como programação inteira.

Finalmente, para ser possível possível resolver qualquer um destes problemas é necessário codificar as variáveis a_i cujos valores vão ser elementos do espaço métrico (M, d) e também codificar a distância d de tal modo que as expressões $d(e_j, a_i)$ sejam lineares nos a_i.

Nesta fase entra-se em conta com as características particulares do espaço métrico onde o problema se coloca. Para ver como se codificam estes espaço vamos considerar duas abordagens possíveis e os casos particulares a elas associadas.

- Numa primeira abordagem o espaço métrico (M, d) é escolhido criteriosamente de modo a ser possível codificar a distância $d(e, a)$ como uma função linear em cada um dos seus argumentos.
  Esta exigência impõe que só para tais espaços métricos é possível codificar os problemas dos K-centros em programação inteira quadrática.

- Na segunda abordagem pretende-se dar maior flexibilidade aos espaços métrico onde o problema dos K-centros possa ser resolvido por programação inteira.
  Ao invés força-se a que todos os centros assumam valores num dado conjunto finito C de pontos de M. Frequentemente esse conjunto de pontos coincide com o *dataset* E, o que implica que cada centro é forcosamento uma experiência, mas pode também ser um sub-conjunto arbitrário, mas finito, de M.
  Nesta abordagem, como iremos ver, a codificação acaba por ser independente do espaço métrico, das experiências concretas E e dos eventuais centros C; a única informação necessária são as cardinalidades deste conjuntos e o conteúdo de uma matriz de distâncias $d(e_j, a_i)$. De fato desde que esta matriz verifique os axiomas na definição distância, pode-se construir uma solução abstraindo tudo o resto.

1ª abordagem

Vamos tomar, para espaço métrico (M, d) o espaço das palavras de n bits, B_n, equipado com a distância de Hamming

$$d(x, y) \equiv \mid \{ i \in 1..n \mid x_i \neq y_i \} \mid$$

É simples verificar que esta distância pode ser escrita como

$$d(x,y) \;=\; \sum_{i=1}^{n} x_i \cdot (1-y_i) + y_i \cdot (1-x_i) \;=\; \sum_{i=1}^{n} x_i + y_i - 2\,x_i\,y_i$$

que é uma expressão linear em cada um dos seus argumentos individuais x ou y, mas é quadrática no par (x,y).

Com este espaço métrico o problema de programação inteira (na versão "minimização") usa:

- Um *dataset* $\{e_j\}_{j\in 1..M}$ formado por M palavras de n bits. Este conjunto pode ser descrito por uma matriz booleana E, de dimensões $M \times N$, tal que a i-ésima componente da palavra e_j é a constante booleana $E_{j,i}$.
- Uma matriz A de dimensões $K \times N$ de *variáveis* booleanas; os K centros a_k são associados às K linhas desta matriz. A i-ésima coluna de A, fornece a componente $a_{k,i}$ de cada um desses centros.
- Uma matriz X de dimensões $K \times M$ de *variáveis* boolenas que descrevem os "clusters" dos centros ou, o que é equivalente, o centro mais próximo de cada experiência.
- Uma variável inteira positiva t

O problema de programação inteira é

$$\text{minimizar } t \quad \text{tal que}$$

$$\forall\, j \in 1..M \;\cdot\quad \sum_{k\in 1..K} X_{k,j} \;=\; 1$$

$$\forall\, j \in 1..M \;\cdot\quad \sum_{k\in 1..K} X_{k,j} \cdot \left(\sum_{i\in 1..n} A_{k,i} + E_{j,i} - 2 \cdot A_{k,i} \cdot E_{j,i}\right) \;\le\; t$$

Note-se que as matrizes E, A, X têm todas componentes booleanas. As matrizes A e X são matrizes de variáveis ; a segunda restrição introduz multiplicações de variáveis $X_{k,j} \cdot A_{k,i}$; tais multiplicações fazem com que este problema vá cair na classe da programação inteira quadrática. Infelizmente este tipo de problema não é suportado por nenhum dos "solvers" usados neste curso.

2ª abordagem

Suponhamos que existem N valores possíveis para os centros e que

$$\mathrm{C} \;\equiv\; \{c_1\,,\, c_2\,,\, \cdots\, c_N\}$$

é uma enumeração desses valores. Como esses valores são conhecidos é possível calcular *a priori* uma matriz constante de distâncias; para $i \in 1..N$ e $j \in 1..M$

$$D_{i,j} \equiv d(c_i, e_j)$$

De facto nem sequer é necessário conhecer o espaço métrico (M, d), o d*ataset* E ou o universo dos centros C; para a resolução do problema basta conhecer os parâmetros M, N e a matriz de distâncias D. O limite nas distâncias t e o número de centros k podem funcionar como parâmetros ou como variáveis a optimizar.

Agora, como o conjunto dos centros A está contido no conjunto de potenciais centros C, vamos representar A por uma variável booleana A de dimensão N com a interpretação de

$$\forall\, i \in 1..N \cdot \quad A_i = 1 \quad \text{sse} \quad c_i \text{ é um centro}$$

Para codificar esta representação dos centros temos de impor a restrição

$$\sum_{i=1}^{N} A_i = k$$

que estabelece a condição de que existem exatamente k centros.

Acrescenta-se a matriz de alocação X que vimos na codificação genérica do problema. A restrição (a) aí definida, que transmite a ideia de unicidade na escolha do "centro mais próximo" é

$$\forall\, j \in 1..M \cdot \quad \sum_{i \in 1..N} X_{i,j} = 1$$

Agora o índice i percorre os potenciais centros e não os centros concretos como vimos na formalização genérica anterior. Por isso se $X_{i,j}$ for igual a 1 isso indica que c_i é um centro concreto; daqui resulta uma nova restrição

$$\forall\, j \in 1..M \cdot \forall\, i \in 1..N \cdot \quad X_{i,j} \leq A_i$$

Agora a restrição (b) , que impõe limites às distâncias, escreve-se facilmente como

$$\forall\, j \in 1..M \cdot \quad \sum_{i \in 1..N} X_{i,j} \cdot D_{i,j} \leq t$$

Nesta expressão o vetor de alocação A identifica os centros, e a matriz de alocação X identifica o centro mais próximo de cada experiência; a matriz D é uma matriz constante que fornece as distâncias entre os eventuais centros e as experiências. Por isso, ao contrário do problema da 1ª abordagem este é um problema de programação linear inteira.

Aqui é possível duas estratégias de optimização:

i. Tomar k como um parâmetro do problema e minimizar o valor da variável *t*.
ii. Tomar t como parâmetros e minimizar o valor de k representado por uma variável.

A escolha de objetivos de optimização torna-se aqui possível porque a restrições são lineares tanto em relação a t como em relação a k.

Como dissemos antes neste problema só interessam os parâmetros M, N e a matriz de distâncias D. Qualquer informação dobre o espaço métrico, os valores concretos das experiências ou dos eventuais centros é irrelevante. Neste contexto, este quatro conjuntos de restrições resolvem completamente o nosso problema.

Como referimos atrás, nesta 2ª abordagem o problema codifica-se em programação inteira linear, o que é uma vantagem em relação à 1ª abordagem. No entanto a matriz de alocação X é muito maior, porque tem uma linha para cada *potencial* centro; ao invés na 1ª abordagem a matriz X tem uma linha para cada centro *concreto*.

Como o número de variáveis X é bastante maior (a não ser que o conjunto C seja muito pequeno) a complexidade do problema linear pode não ser muito diferente da complexidade do problema usado na 1ª abordagem.