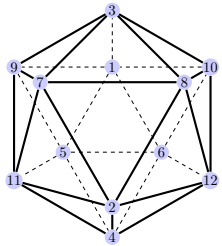
Mathheuristics 11 - Heuristicas e Solvers

Alexandre Checoli Choueiri

09/11/2025



Definição e Classificação

Introdução

Sabemos que a base de todo problema de otimização é um modelo matemático. Mesmo que não consigamos resolvê-lo por métodos exatos (Branch and Bound, Branch and Cut, etc...), **a mera criação do modelo nos ajuda a compreender melhor as complexidades do problema**. No entanto, com o modelo criado e implementado (em um *solver*), existem muitas possibilidades de **colaboração** entre o mesmo e metaheuristicas, a fim de melhorar o processo de busca - essa colaboração tem o nome de **Mathheuristics** (Heurísticas Matemáticas).

Definição

Vejamos a definição de *Mathheuristics*, fornecida por *Boschetti e Maniezzo - Contemporary approaches in matheuristics an updated survey*:

Definição

Mathheuristics: são algoritmos (*frameworks*) independentes de problemas, que usam programação matemática para obter soluções heuristicas de alta qualidade.

Definição

Vejamos a definição de *Mathheuristics*, fornecida por *Boschetti e Maniezzo - Contemporary approaches in matheuristics an updated survey*:

Definição

Mathheuristics: são algoritmos (*frameworks*) independentes de problemas, que usam programação matemática para obter soluções heuristicas de alta qualidade.

Ou seja, elas são:

1. **Independentes de problemas**: como as meta-heurísticas, devem ser *frameworks* genéricos.
2. **Obtém soluções heurísticas**: ainda estamos em métodos não-exatos.

Uma palavra de cautela

Meu objetivo com a aula é mostrar diversas possibilidades de utilizar o *solver* com meta-heurísticas - não necessariamente somente os métodos genéricos, então alguns dos tópicos aqui não seriam considerados Mathheuristics pela definição de Boschetti. Eu proponho a seguinte separação:

Uma palavra de cautela

Meu objetivo com a aula é mostrar diversas possibilidades de utilizar o *solver* com meta-heurísticas - não necessariamente somente os métodos genéricos, então alguns dos tópicos aqui não seriam considerados Mathheuristics pela definição de Boschetti. Eu proponho a seguinte separação:

1. **Interação Indireta:** como podemos tentar ajudar o solver (e sermos ajudados por ele), de uma forma não tão direta?

Uma palavra de cautela

Meu objetivo com a aula é mostrar diversas possibilidades de utilizar o *solver* com meta-heurísticas - não necessariamente somente os métodos genéricos, então alguns dos tópicos aqui não seriam considerados Mathheuristics pela definição de Boschetti. Eu proponho a seguinte separação:

1. **Interação Indireta:** como podemos tentar ajudar o solver (e sermos ajudados por ele), de uma forma não tão direta?
2. **Nas entranhas do Solver:** como podemos usar somente o Solver para encontrar soluções? (definição de *Mathheuristics*)

Uma palavra de cautela

Meu objetivo com a aula é mostrar diversas possibilidades de utilizar o *solver* com meta-heurísticas - não necessariamente somente os métodos genéricos, então alguns dos tópicos aqui não seriam considerados Mathheuristics pela definição de Boschetti. Eu proponho a seguinte separação:

1. **Interação Indireta:** como podemos tentar ajudar o solver (e sermos ajudados por ele), de uma forma não tão direta?
2. **Nas entranhas do Solver:** como podemos usar somente o Solver para encontrar soluções? (definição de *Mathheuristics*)
3. **Decomposições:** como decompor problemas pode nos permitir usar modelos matemáticos?

Uma palavra de cautela

Meu objetivo com a aula é mostrar diversas possibilidades de utilizar o *solver* com metaheurísticas - não necessariamente somente os métodos genéricos, então alguns dos tópicos aqui não seriam considerados Mathheuristics pela definição de Boschetti. Eu proponho a seguinte separação:

1. **Interação Indireta:** como podemos tentar ajudar o solver (e sermos ajudados por ele), de uma forma não tão direta?
2. **Nas entranhas do Solver:** como podemos usar somente o Solver para encontrar soluções? (definição de *Mathheuristics*)
3. **Decomposições:** como decompor problemas pode nos permitir usar modelos matemáticos?
4. **Sub-modelos:** como usar o solver para resolver subproblemas em Metaheuristicas?

Uma palavra de cautela

Meu objetivo com a aula é mostrar diversas possibilidades de utilizar o *solver* com metaheurísticas - não necessariamente somente os métodos genéricos, então alguns dos tópicos aqui não seriam considerados Mathheuristics pela definição de Boschetti. Eu proponho a seguinte separação:

1. **Interação Indireta:** como podemos tentar ajudar o solver (e sermos ajudados por ele), de uma forma não tão direta?
2. **Nas entranhas do Solver:** como podemos usar somente o Solver para encontrar soluções? (definição de *Mathheuristics*)
3. **Decomposições:** como decompor problemas pode nos permitir usar modelos matemáticos?
4. **Sub-modelos:** como usar o solver para resolver subproblemas em Metaheuristicas?
5. **Restrições preguiçosas:** como inserir restrição somente quando são necessárias?

Interação Indireta

Interação Indireta

Relembrando o Bound

Vamos tomar como *interação indireta* uma combinação mais branda entre a Metaheuristica e o solver, somente trocando informações sobre as soluções. Para entender essa interação, é necessário relembrar o algoritmo *Branch and Bound*, principalmente como funciona o *Bound*. Para isso veja a apresentação - principalmente a parte final, em que o tamanho de uma árvore é determinado pela determinação do *Bound*.

Interação Indireta

Retroalimentação

Com isso em mente, podemos usar as 2 seguintes ideias com o solver-meta (uma sequencial e uma em paralelo):

[1] Esse método é mais complexo, pois envolve a implementação em *threads* para conseguir o paralelismo

Interação Indireta

Retroalimentação

Com isso em mente, podemos usar as 2 seguintes ideias com o solver-meta (uma sequencial e uma em paralelo):

1. **Sequencial :** rodamos a metaheuristica qualquer e obtemos uma solução. Em seguida alimentamos essa solução inicial factível para o solver: assim ele já começa com um Bound de uma sol. factível.

[1] Esse método é mais complexo, pois envolve a implementação em *threads* para conseguir o paralelismo

Interação Indireta

Retroalimentação

Com isso em mente, podemos usar as 2 seguintes ideias com o solver-meta (uma sequencial e uma em paralelo):

1. **Sequencial :** rodamos a metaheuristica qualquer e obtemos uma solução. Em seguida alimentamos essa solução inicial factível para o solver: assim ele já começa com um Bound de uma sol. factível.
2. [1]**Paralelo :** iniciamos os processos em paralelo, tanto o solver quanto a metaheuristica, e sempre que um dos dois encontrar uma solução melhor, ele alimenta o outro processo.

[1]Esse método é mais complexo, pois envolve a implementação em *threads* para conseguir o paralelismo

Nas entranhas do Solver

Nas entranhas do Solver

Existem duas heurísticas genéricas clássicas que são usadas somente com o Solver (essas são de fato Mathheuristics!). Uma delas serve para **criar uma solução inicial**, e a outra serve para **melhorar uma solução já existente**. Ambas são baseadas na fixação de variáveis do modelo (tratar as mesmas como se fossem parâmetros).

Nas entranhas do Solver

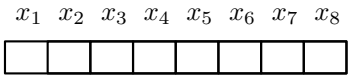
Existem duas heurísticas genéricas clássicas que são usadas somente com o Solver (essas são de fato Mathheuristics!). Uma delas serve para **criar uma solução inicial**, e a outra serve para **melhorar uma solução já existente**. Ambas são baseadas na fixação de variáveis do modelo (tratar as mesmas como se fossem parâmetros).

1. **Relax-and-fix**: Heurística baseada em relaxar as variáveis inteiras do modelo de forma iterativa, até que uma solução inteira factível seja encontrada - encontra uma solução inicial.

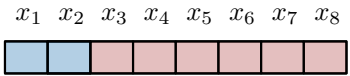
Nas entranhas do Solver

Existem duas heurísticas genéricas clássicas que são usadas somente com o Solver (essas são de fato Mathheuristics!). Uma delas serve para **criar uma solução inicial**, e a outra serve para **melhorar uma solução já existente**. Ambas são baseadas na fixação de variáveis do modelo (tratar as mesmas como se fossem parâmetros).

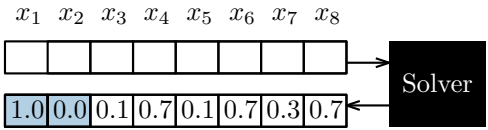
1. **Relax-and-fix**: Heurística baseada em relaxar as variáveis inteiras do modelo de forma iterativa, até que uma solução inteira factível seja encontrada - encontra uma solução inicial.
2. **Fix-and-optimize**: Fixa um conjunto de variáveis e tenta otimizar o modelo com o resto.



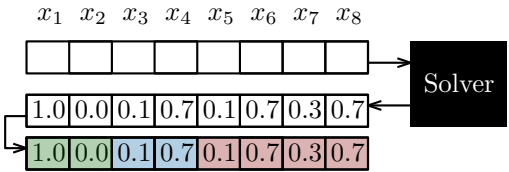
Suponha um problema de otimização com 8 variáveis que devem ser **inteiras**. Determinamos um conjunto de variáveis que serão relaxadas, ou seja, mantidas como contínuas.



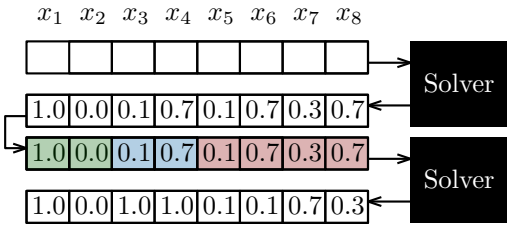
Supondo x_1 e x_2 inteiras e as outras relaxadas.



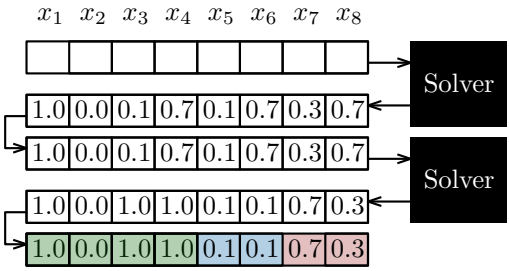
Usamos esse modelo relaxado no solver. Após a otimização coletamos o resultado - agora já temos 2 **variáveis inteiras**.



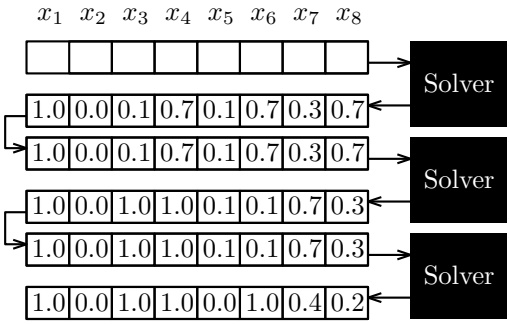
Na próxima iteração, para não perder os valores inteiros encontrados, fixamos os mesmos - esses não podem ser alterados no processo de otimização (como se fossem parâmetros), selecionamos outros para ficarem agora inteiros e o resto fica relaxado.



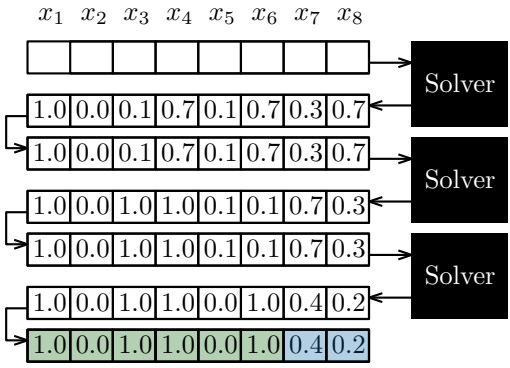
E novamente chamamos o *solver*.



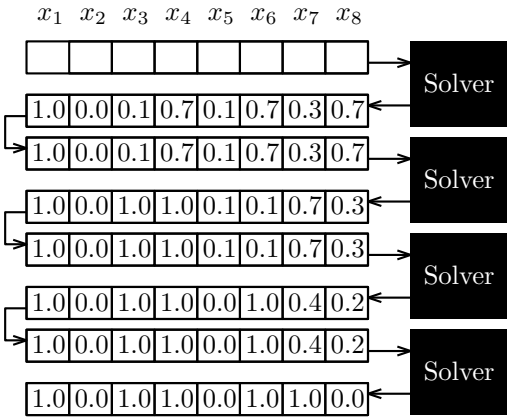
Na próxima iteração temos 4 variáveis inteiras.



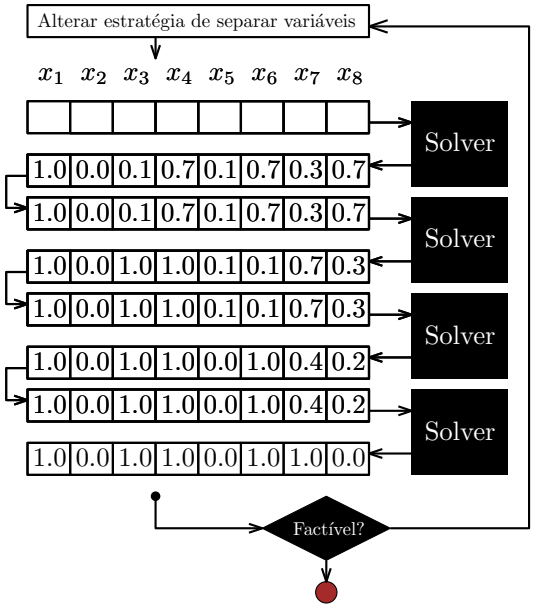
O processo é repetido até que **todas as variáveis tenham sido integralizadas**.



O processo é repetido até que **todas as variáveis tenham sido integralizadas**.



O processo é repetido até que **todas as variáveis tenham sido integralizadas**.



No fim, pode acontecer da solução ser **infactível**. Nesse caso o processo todo deve recomeçar, alterando a estratégia de separação de variáveis inteiras x relaxadas.

Relax-and-Fix

Algumas **observações**:

1. **Relax-and-fix**: O nome do método é dado devido ao processo iterativo de relaxar e fixar as variáveis do problema. Note que é independente do problema - a única escolha é a estratégia de separação de variáveis.

Relax-and-Fix

Algumas **observações**:

1. **Relax-and-fix**: O nome do método é dado devido ao processo iterativo de relaxar e fixar as variáveis do problema. Note que é independente do problema - a única escolha é a estratégia de separação de variáveis.
2. Esse é um método de **construção de solução**, e pode não conseguir gerar uma solução factível. Pode-se repetir o mesmo um número de vezes, sempre alterando o critério de separação das variáveis.

Relax-and-Fix

Algumas **observações**:

1. **Relax-and-fix**: O nome do método é dado devido ao processo iterativo de relaxar e fixar as variáveis do problema. Note que é independente do problema - a única escolha é a estratégia de separação de variáveis.
2. Esse é um método de **construção de solução**, e pode não conseguir gerar uma solução factível. Pode-se repetir o mesmo um número de vezes, sempre alterando o critério de separação das variáveis.
3. Como a resolução de PLs é mais fácil do que de PIs, usualmente relaxando e fixando variáveis deixa a resolução do modelo mais rápida, o que permite a iteratividade do algoritmo Relax-and-Fix.

Fix-and-optimize

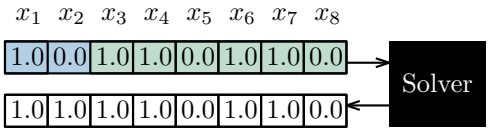
A heurística **Fix-and-Optimize** é uma heurística de melhoria - ou seja, precisa de uma solução inicial, mas segue uma ideia parecida à *Relax-and-Fix*. Ele já possui uma solução inicial factível, e a cada iteração, escolhe um conjunto de variáveis para serem fixadas, e novamente usa o solver para tentar otimizar as outras (elas mantém a restrição de integralidade). O processo se repete por um número definido de iterações.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0

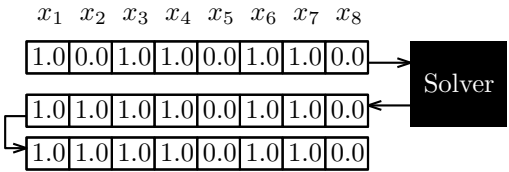
Considere novamente um problema com 8 variáveis, em que **já temos uma solução factível inteira** (não ótima). Queremos tentar melhorá-la.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0

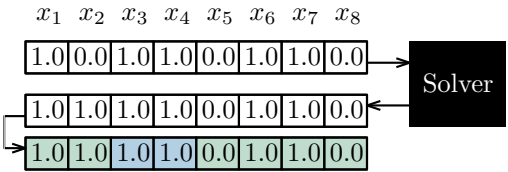
Escolhemos um conjunto de variáveis que serão fixadas, e deixamos o solver tentar otimizar o valor das outras. Como fixamos algumas variáveis, o tempo de processamento fica muito menor, dado que para o solver **elas são consideradas como parâmetros** (não tem como serem alteradas).



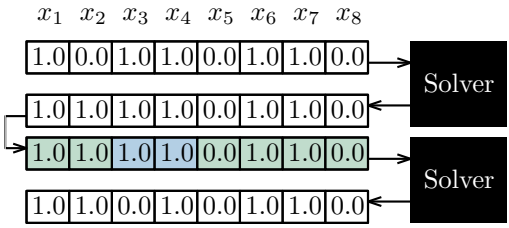
Com esse modelo usamos o solver novamente, na esperança de que alguma alteração nas variáveis não relaxadas possa melhorar a função objetivo



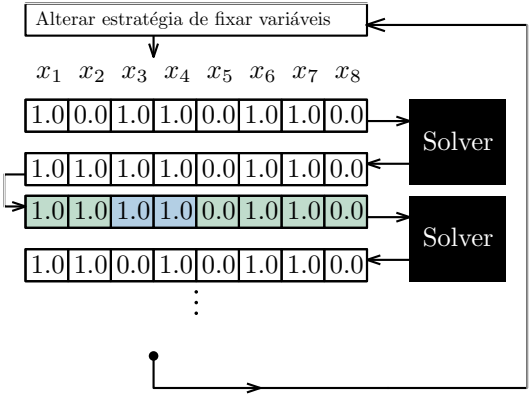
Com a nova solução, escolhemos outro conjunto de variáveis para serem fixadas, e outro para deixar livre.



E o processo se repete, até todas as variáveis terem sido fixadas...



E o processo se repete, até todas as variáveis terem sido fixadas...



Ao fim, pode-se reiniciar o processo, escolhendo outra estratégia para fixar variáveis.

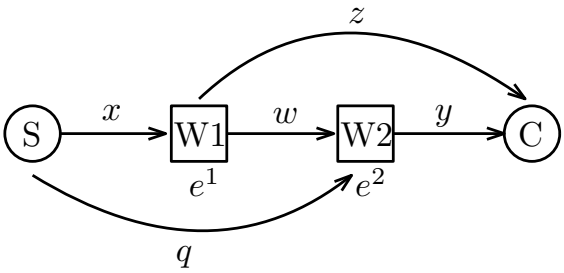
Fix-and-optimize e Relax-and-Fix

Percebemos então que ambas as heurísticas, **Relax-and-Fix** e **Fix-and-Optimize** usam fortemente o solver de maneira iterativa. E também são templates genéricos para quaisquer problemas - bastando definir a estratégia de fixação e relaxação das variáveis. Dessa forma, ambas podem ser consideradas como *Mathheuristics* pela nossa definição.

Vejamos um exemplo em que o F&O é utilizado, e o próprio método para escolha de variáveis fixadas é uma meta-heurística.

Fix-and-Optimize

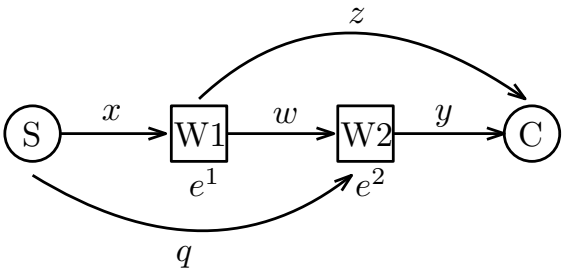
Malha Logística



Considere o problema de determinar quantidades de produtos que devem sair de depósitos (**S**) para atender a demandas de clientes (**C**). As quantidades podem ser enviadas a depósitos (**W1**, **W1**), que podem armazenar as produtos em estoques. Os depósitos possuem um custo fixo de abertura.

Fix-and-Optimize

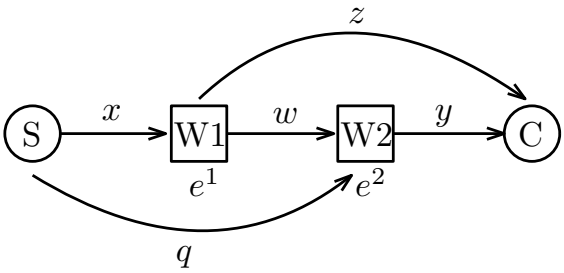
Malha Logística



O modelo completo é muito complexo para ser resolvido. No entanto, **se os depósitos que forem usados já estiverem determinados (fixados)**, o modelo é resolvido de forma mais simples. Mas como determinar quais depósitos devem ser abertos?

Fix-and-Optimize

Malha Logística



Neste caso, podemos usar uma meta-heurística para determinar somente os depósitos que são abertos ou fechados. A cada iteração, o modelo completo com esses depósitos fixados é executado, para que se obtenha o valor da função objetivo.

Decomposições

Decomposições

Em muitos casos, usar um modelo matemático completo para o problema é **inviável**, no entanto, podemos realizar decomposições no mesmo, resolvendo partes menores e realimentando outros modelos com os resultados anteriores. Nem todos os problemas permitem tal abordagem, por isso é importante conhecer bem a sua estrutura, e saber identificar partes que podem ser desacopladas.

Decomposições

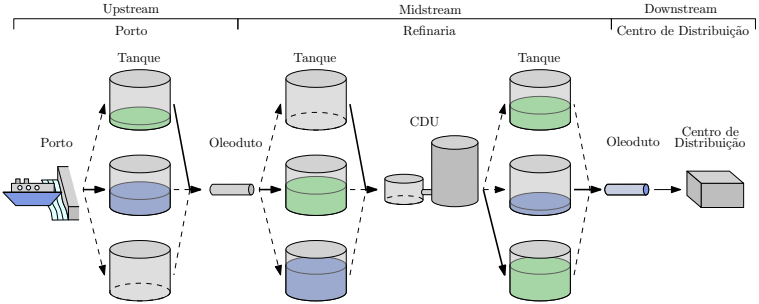
Em muitos casos, usar um modelo matemático completo para o problema é **inviável**, no entanto, podemos realizar decomposições no mesmo, resolvendo partes menores e realimentando outros modelos com os resultados anteriores. Nem todos os problemas permitem tal abordagem, por isso é importante conhecer bem a sua estrutura, e saber identificar partes que podem ser desacopladas.

1. Sequenciamento em Oleodutos
2. Balancemento de linha

Decomposições

Sequenciamento de Petróleo

No problema de sequenciamento de petróleo completo, precisamos determinar as quantidades/fluxos/etc...que saem dos campos de petróleo cru e chegam nas refinarias, bem como as quantidades de produtos beneficiados que saem das refinarias e chegam aos clientes.



Decomposições

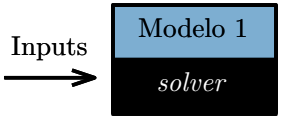
Sequenciamento de Petróleo

Uma abordagem para resolver o problema é **decompor o mesmo em 2 partes**: um modelo que determina as quantidades campos-refinarias, e outro, que usa as respostas do antigo como inputs, que determinam as quantidades refinaria-clientes.

Decomposições

Sequenciamento de Petróleo

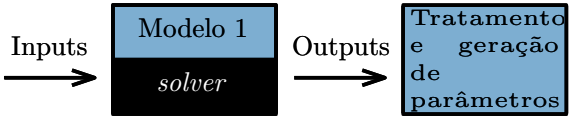
Uma abordagem para resolver o problema é **decompor o mesmo em 2 partes**: um modelo que determina as quantidades campos-refinarias, e outro, que usa as respostas do antigo como inputs, que determinam as quantidades refinaria-clientes.



Decomposições

Sequenciamento de Petróleo

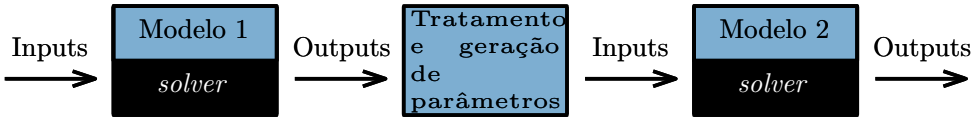
Uma abordagem para resolver o problema é **decompor o mesmo em 2 partes**: um modelo que determina as quantidades campos-refinarias, e outro, que usa as respostas do antigo como inputs, que determinam as quantidades refinaria-clientes.



Decomposições

Sequenciamento de Petróleo

Uma abordagem para resolver o problema é **decompor o mesmo em 2 partes**: um modelo que determina as quantidades campos-refinarias, e outro, que usa as respostas do antigo como inputs, que determinam as quantidades refinaria-clientes.



Decomposições

Balanceamento de Linha/ Sequenciamento

...

Solver dentro das MH

Decomposições

Algumas vezes, ao resolver um problema grande, nos deparamos com sub-problemas menores (já conhecidos) que podem ser resolvidos de forma exata por *solvers*.

1. VRP - Determinação de combinação de rotas por particionamento de conjuntos.
2. VRP - Bin Packing para minimizar o número de veículos ao criar os grupos de rotas.

Restrições Preguiçosas (*Lazy Constraints*)

Lazy Constraints

As *lazy-constraints* são restrições que vão sendo adicionadas ao modelo **conforme forem sendo necessárias** (*on demand*). Essa técnica é útil quando se trabalha com modelos em que o número de restrições cresce muito rapidamente em função do tamanho das instâncias

1. Problemas de roteirização: restrições para impedimento de sub-rotas.

Lazy Constraints

Considere o modelo matemático para o TSP de Dantzig-Fulkerson:

$$\text{Minimizar:} \quad \sum_{i \in N} \sum_{j \in N, j \neq i} c_{ij} x_{ij} \tag{1}$$

$$\text{sujeito a:} \quad \sum_{j \in N, j \neq i} x_{ij} = 1, \qquad \forall i \in N \tag{2}$$

$$\sum_{i \in N, i \neq j} x_{ij} = 1, \qquad \forall j \in N \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S, j \neq i} x_{ij} \leq |S| - 1, \quad \forall S \subset N,\ 2 \leq |S| \leq |N| - 1 \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in N,\ i \neq j \tag{5}$$

Lazy Constraints

A restrição de eliminação de sub-rotas cresce de forma exponencial com o número de pontos.

$$\text{Minimizar:} \quad \sum_{i \in N} \sum_{j \in N, j \neq i} c_{ij} x_{ij} \tag{1}$$

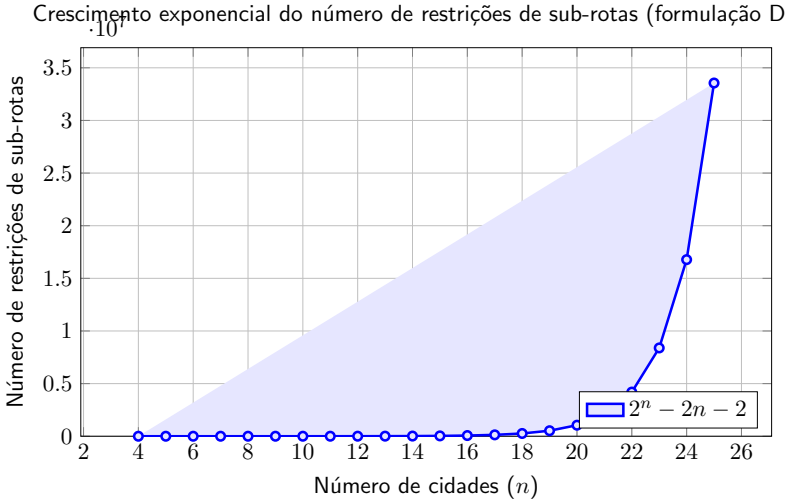
$$\text{sujeito a:} \quad \sum_{j \in N, j \neq i} x_{ij} = 1, \qquad \forall i \in N \tag{2}$$

$$\sum_{i \in N, i \neq j} x_{ij} = 1, \qquad \forall j \in N \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S, j \neq i} x_{ij} \leq |S| - 1, \quad \forall S \subset N,\ 2 \leq |S| \leq |N| - 1 \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in N,\ i \neq j \tag{5}$$

Lazy Constraints)



Lazy Constraints

Nesse caso, podemos resolver o **modelo relaxado**: removendo as restrições de subrotas - o modelo de designação.

$$\text{Minimizar:} \quad \sum_{i \in N} \sum_{j \in N, j \neq i} c_{ij} x_{ij} \tag{1}$$

$$\text{sujeito a:} \quad \sum_{j \in N, j \neq i} x_{ij} = 1, \quad \forall i \in N \tag{2}$$

$$\sum_{i \in N, i \neq j} x_{ij} = 1, \quad \forall j \in N \tag{3}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i, j \in N,\ i \neq j \tag{5}$$

Lazy Constraints

Em seguida, de forma iterativa, o modelo é:

1. Resolvido.
2. A resposta é verificada para identificar sub-rotas.
3. Se existirem subrotas: criar as restrições somente para elas e **voltar para 1**, senão, FIM.